1985-12

Development of a testbed for multisensor distributed decision algorithms.

Schon, Mark Alan

http://hdl.handle.net/10945/21274

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

DEVELOPMENT OF A TESTBED FOR MULTISENSOR DISTRIBUTED DECISION ALGORITHMS

By
Mark A. Schon

December 1985

Thesis Advisor: Charles W. Therrien

Approved for public release; distribution is unlimited.

ːURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

. SECURITY CLASSIFICATION AUTHORITY

3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited

. DECLASSIFICATION/DOWNGRADING SCHEDULE

PERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)

. NAME OF PERFORMING ORGANIZATION: aval Postgraduate chool

6b. OFFICE SYMBOL (If applicable): 62

7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School

ADDRESS (City, State, and ZIP Code): onterey, CA 93943-5100

7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5100

. NAME OF FUNDING/SPONSORING ORGANIZATION

8b. OFFICE SYMBOL (If applicable)

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

. ADDRESS (City, State, and ZIP Code)

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

TITLE (Include Security Classification): EVELOPMENT OF A TESTBED FOR MULTISENSOR DISTRIBUTED ECISION ALGORITHMS

PERSONAL AUTHOR(S): Schon, Mark A.

a TYPE OF REPORT: aster's Thesis

13b TIME COVERED: FROM ______ TO ______

14. DATE OF REPORT (Year, Month, Day): 1985 December

15. PAGE COUNT: 85

. SUPPLEMENTARY NOTATION

COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Distributed Decision Processing; Computer Network; Microcomputer Clusters; Process Synchronization; Network Communication

ABSTRACT (Continue on reverse if necessary and identify by block number)

Distributed decision problems arise when two or more sensors viewing he same phenomenon must work cooperatively to draw inferences about the bserved situation. Typical examples are in target detection and target lassification. Such problems are characterized by distributed processing f information and communication between processors over a limited andwidth data link. This thesis presents some statistical distributed ecision algorithms and describes the implementation of one of them on a et of loosely coupled multiprocessor clusters which simulate the istributed environment characterizing multisensor decision problems.

). DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED

a. NAME OF RESPONSIBLE INDIVIDUAL: C. W. Therrien

22b TELEPHONE (Include Area Code): (408)646-2160

22c. OFFICE SYMBOL: 62Ti

D FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

The purpose of the implementation was to investigate problems of communication and process synchronization in a pair of processor clusters performing a statistical distributed decision algorithm. This thesis describes how these communication and synchronization problems were addressed and solved.

Development of a Testbed for Multisensor Distributed Decision Algorithms

by

Mark Alan Schon
Captain, United States Marine Corps
B.S., University of Utah, 1976

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1985

ABSTRACT

Distributed decision problems arise when two or more sensors viewing the same phenomenon must work cooperatively to draw inferences about the observed situation. Typical examples are in target detection and target classification. Such problems are characterized by distributed processing of information and communication between processors over a limited bandwidth data link. This thesis presents some statistical distributed decision algorithms and describes the implementation of one of them on a set of loosely coupled multiprocessor clusters which simulate the distributed environment characterizing multisensor decision problems. The purpose of the implementation was to investigate problems of communication and process synchronization in a pair of processor clusters performing a statistical distributed decision algorithm. This thesis describes how these communication and synchronization problems were addressed and solved.

DISCLAIMER

Some terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each occurrence of a trademark, all trademarks appearing in this thesis are listed below following the name of the firm holding the trademark:

1. INTEL Corporation, Santa Clara, California

   8086 MULTIBUS

2. Digital Research, Pacific Grove, California

   PL/I-86 LINK86

3. XEROX Corporation, Stamford, Connecticut

   Ethernet Local Area Network

4. InterLAN Corporation, Westford, Massachusetts

   NI3010 Ethernet Communication Controller Board

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. PROBLEM DESCRIPTION

Modern military battle systems increasingly rely on the coordinated use of information from multiple sources to assess the battlefield situation. Two or more remotely located sensors may observe the same object with the purpose of drawing inferences about the observation. A common example is in the use of radars to detect and eventually classify objects for purposes of an appropriate response. In this type of scenario it is important to process the acquired information jointly to arrive at the optimum or near optimum decision.

A simple example to demonstrate the distributed decision scenario is illustrated in Fig. 1 and explained below.

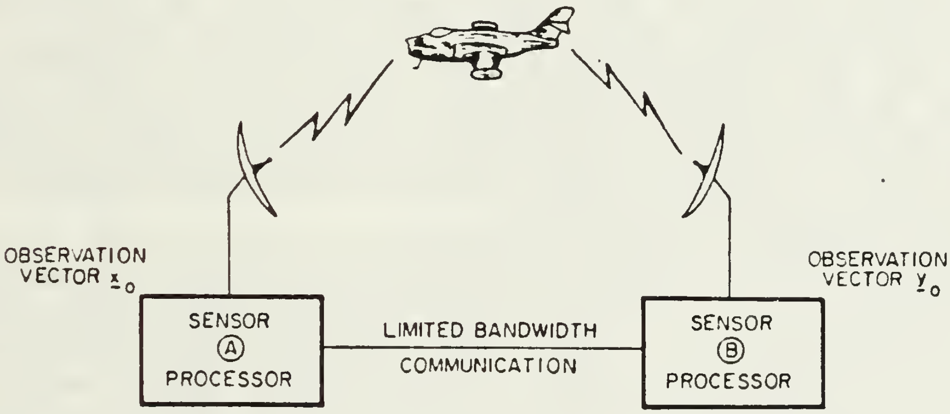

Figure 1 - Distributed Decision Scenario

Two sensors, labeled **A** and **B**, observe the same area in space to jointly make a binary decision based on the statistical properties of the observations: either a target is detected or there is no target detected. In certain situations the optimal decision made by each sensor acting individually would result in each deciding that a target exists when an optimal joint decision would decide that a target does not exist. This dichotomy points out that in order for a higher level process

to make a correct decision about the object, the distributive nature of the problem must be built into the front end statistical decision procedure.

The problem of distributed processing of the observation data to achieve optimal or near optimal decisions is discussed herein. The sensors are configured to perform computations to reduce the observation data and to communicate among themselves over a limited bandwidth channel. Algorithms which operate in this environment are called distributed decision algorithms. Algorithms which perform computations on all the observation data collected and gathered at one central location are called centralized algorithms.

This thesis deals specifically with modeling a particular class of distributed decision algorithms in a multiprocessor environment, and with related issues of process synchronization. Although centralized algorithms are not of concern here, a companion thesis [1] deals with non real-time simulation and evaluation of distributed decision algorithms and comparison with centralized algorithms.

Although the thesis deals with a particular class of distributed decision algorithms, the implementation problems presented by the algorithm would be typical of most distributed algorithms. Thus the work can be regarded as developing a test facility in which distributed decision algorithms can be tested in a realistic computational environment.

The problem is to model the processing environment of two sensors which collect data on a common object. The sensors and their associated processors then perform parallel processing to partially reduce the data and the partial results are exchanged via a local area network. A final decision about the observed object is then made at each sensor, based on the locally processed data and the exchanged information.

B. HARDWARE/SOFTWARE CONFIGURATION

The hardware/software configuration used for the modeling of the distributed decision network was the REAL-TIME CLUSTER STAR (RTC *) system. This system was developed by thesis students under the AEGIS Project Group at the Naval Postgraduate School. RTC * was designed to handle algorithms of the type

incorporated here with the appropriate synchronization and control primitives. The hardware consists of two clusters of single board computers (SBC's) sharing a common backplane with an Ethernet local area network(**LAN**) serving as the communications link. The operating system is a distributed multicomputer real-time executive that permits asynchronous parallel operation of processes resident on SBC's of the same cluster and in separate clusters linked by the LAN. User processes, such as the distributed decision algorithms, are resident in the local memory of each SBC. They can share data and control variables using the common memory in each cluster, as well as the backplane and the LAN data paths.

A detailed description of the hardware system and the software operating system is provided in [2]. The distributed decision algorithms are organized as a number of separate processes on various single board microcomputers in the two cluster arrangement. Process synchronization is achieved through certain **await**, **advance**, and **read** primitives to control the orderly multiple/parallel process execution as well as a sequencer to control the allocation of the LAN shared resource. Each cluster simulates the operations that would be performed by the sensor processors. Data read from disk storage simulates the input sensor observations. Each processor then performs the necessary computations to reduce the data to the statistics required for the joint decision. One set of statistics are then exchanged between clusters while another set is retained locally and computation is continued to produce a combined statistic based on the joint data. This combined statistic is then compared with a predetermined threshold to make the detection decision. Computations continue while data is available for input and the decision results are displayed on a local console of each cluster.

C. STRUCTURE OF THE THESIS

In the remainder of this thesis the distributed decision problem is defined and various distributed decision algorithms and their characteristics are discussed. The implementation of one algorithm in a distributed multiprocessor test environment is introduced and discussed in detail. Emphasis is placed on

obtaining solutions to the problems of communication and synchronization for processes operating in two remote computer systems. The specific contents of each chapter is as follows.

Chapter II presents the distributed decision problem with a discussion of a specific distributed decision algorithm. Simple examples illustrate the detection problem with a binary decision rule.

Chapter III presents the implementation of a specific detection distributed decision algorithm in the RTC* multicomputer system and discusses important issues relevant to the implementation of this type of algorithm.

Chapter IV is a summary of the findings and summarizes the results of the implementation in the RTC* multicomputer system.

II. DISTRIBUTED DECISION ALGORITHMS

A. SUMMARY OF ALGORITHMS

Alternative approaches to a simple binary (two hypothesis) decision problem are presented in this chapter. The various algorithms have overall similar characteristics in that local computations are performed by each sensor, reduced data is exchanged over a limited capacity communications channel, and final decisions are made based on the joint observations of the sensors.

The discussion here assumes that there are only two sensors involved (A and B) and that the task is to make a binary decision (H_1: target is present, or H_2: no target is present). Generalization of most of these methods to multiple sensors and/or multiple hypotheses is possible.

1. Tenney - Sandell Algorithm

Tenney and Sandell [3] seem to have been the first to look at distributed decision algorithms of the type described here. In their work, the observations of the two sensors are assumed to be independent when conditioned on the decision hypotheses. Such independence of observations could arise if the sensors measured different physical properties of the target (e.g. radar cross section and infrared radiation). The sensors each make a binary decision based on their own observations and send the result (a single bit) to a fusion center for arbitration. A cost criterion was devised that depends on the decisions made by each sensor and on the two hypotheses. Tenney and Sandell showed that the procedure that minimized the expected value of the cost is a likelihood ratio test at each sensor. However the thresholds used by the two sensors are coupled through some integral equations.

2. Relaxation Algorithms

Relaxation algorithms [4,5] are another way to execute distributed decisions. These algorithms are less well-founded in a theoretical sense, but seem

to work well in practice. In the relaxation algorithm each sensor makes an initial decision based on its own observations. The decisions are exchanged and each sensor may then revise its decision based on the new information. The procedure works best when there are multiple decision makers involved and may require more than a single iteration to converge.

3. The Generalized Likelihood Ratio Test

If the information exchanged between sensors is more than a single bit, but limited to, say, a single floating point number, then a whole new class of procedures can be suggested. In particular, if the observations are independent as in the Tenney-Sandell analysis, then the likelihood ratio for the joint observations factors into two parts, each depending only on the observations of a single sensor. Thus each sensor can compute the likelihood ratio (or log likelihood ratio) statistic for its own observations and send it to the other sensor. Each sensor then has the complete information required for making a decision to minimize probability of error based on the joint observations.

A more interesting problem occurs if the observations are correlated. In this case the joint likelihood ratio does not factor in such a convenient way. However, a procedure can be suggested that leads to a relatively simple decision algorithm. Let the observations acquired by sensors A and B be represented by $\mathbf{x}_0$ and $\mathbf{y}_0$ respectively. The optimal centralized test to minimize the probability of error has the form

$$\ln\frac{p_1(\mathbf{x}_0,\mathbf{y}_0)}{p_2(\mathbf{x}_0,\mathbf{y}_0)} = \ln\frac{p_1(\mathbf{x}_0)}{p_2(\mathbf{x}_0)} + \ln\frac{p_1(\mathbf{y}_0 \mid \mathbf{x}_0)}{p_2(\mathbf{y}_0 \mid \mathbf{x}_0)} \underset{H_2}{\overset{H_1}{\gtrless}} \ln T \tag{1}$$

where the subscript i on each probability density function p indicates that the density function is for hypothesis H_i. A distributed form of this test can be developed by allowing sensor A to compute the first term in (1) and allowing sensor B to compute an approximation to the second term (the conditional log likelihood ratio) by using some estimate for the observations $\mathbf{x}_0$. This procedure is

known as a *generalized likelihood ratio test* [6]. In essence, when the density function involves an unknown parameter (in our case $\mathbf{x}_0$ in the second term in (1)) estimates are made based on each hypothesis ($\mathbf{x}_1$ for H_1 and $\mathbf{x}_2$ for H_2) and used in the corresponding density function. The form of the second term then becomes

$$\ln \frac{p_1(\mathbf{y}_0 \mid \hat{\mathbf{x}}_1)}{p_2(\mathbf{y}_0 \mid \hat{\mathbf{x}}_2)} \tag{2}$$

If sensor B sends the result of this computation to sensor A, then the test (1), can be evaluated to make a decision. A symmetric computation can be made with the roles of A and B reversed, where the the estimates for $\mathbf{y}_0$ are $\hat{\mathbf{y}}_1$ and $\hat{\mathbf{y}}_2$ at sensor A.

The decision rule just described has a number of essential differences from the corresponding centralized algorithm. First, since the likelihood ratio evaluated by one sensor uses an *estimate* for the other sensor's observations, the performance of the algorithm will in general be different and suboptimal when compared to the centralized test. Second, since the two sensors perform symmetric computations with the roles of $\mathbf{x}_0$ and $\mathbf{y}_0$ reversed, there will, in general be a region of the combined observation space where the decisions of the two sensors do not agree. The properties of this class of distributed decision algorithms is dependent on the various methods of estimating the unknown observations $\mathbf{x}_0$. If the sensors are allowed to exchange only a single statistic then the estimate for $\mathbf{x}_0$ must be derived entirely from $\mathbf{y}_0$ (e.g. using MAP estimation) and the resulting decision rule is of the form

$$\lambda_A(\mathbf{x}_0) + \lambda_B'(\mathbf{y}_0) \underset{H_2}{\overset{H_1}{\gtrless}} \ln T \tag{3}$$

This limits the degree to which the distributed test can approximate the centralized test since in many cases the centralized test will not be separable.

The log likelihood ratios in (3) are computed at their respective sensors and once the primed statistic is received, it is added to the unprimed locally

computed statistic and the result is compared to the known threshold, ln T. For the case of Gaussian observations, the densities, p_1 and p_2 of (1), are of the form

$$p_i = \frac{1}{(2\pi)^{\frac{N}{2}} \, |\mathbf{K}^{(i)}|^{\frac{1}{2}}} \exp\left[-\frac{1}{2} [\mathbf{x} - \mathbf{m}^{(i)}]^T \, [\mathbf{K}^{(i)}]^{-1} \, [\mathbf{x} - \mathbf{m}^{(i)}] \right] \quad i = 1,2 \tag{4}$$

and the points where the sum of the statistics in (3) is equal to ln T establish a decision boundary for this particular decision rule. If observations $\mathbf{x}_0$ and $\mathbf{y}_0$ result in a point with value greater than the boundary value, the decision is H_1 and if the value is less than the boundary value the decision is H_2. Decision boundaries for Gaussian density functions are generally elliptical, parabolic, or hyperbolic and define two (not necessarily connected) regions, one for each hypothesis.

4. Decision Based on the Nearest Neighbor Rule

A final form of distributed decision algorithm is based on the k-nearest neighbor rule of pattern recognition [7]. In this nonparametric decision rule, a set of observations to be tested is represented as a point in a multidimensional observation space. Also existing in this space are previously given sets of points (training data) corresponding to each of the two hypotheses. The distance of the measured observations to each of the other points is computed to determine its k nearest neighbors. If most of the neighbors correspond to H_1 then the given observations are also associated with H_1, otherwise the given observations are classified according to H_2.

A distributed form of this decision rule can be developed by letting each sensor determine a small number of nearest neighbors in the $\mathbf{x}$ or $\mathbf{y}$ subspace. If the labels of these points and their distances from the observation data are interchanged, one can compute the distances in the $\mathbf{xy}$ observation space and classify the observation data. This policy does not guarantee that the true nearest neighbors will always be found but allows a decision to be made without further iterations and exchange of information.

B. GENERALIZED LIKELIHOOD RATIO TEST

The algorithm based on the generalized likelihood ratio test was chosen for implementation on the distributed system. It has requirements for communication and process synchronization that are representative of distributed decision algorithms in general. The performance characteristics of the generalized likelihood ratio test are investigated in [1]. If the joint density function for vector observations $\mathbf{x}$ and $\mathbf{y}$ is Gaussian, then a quadratic decision boundary results. This is known as a quadratic classifier [8]. The joint density function for observations $\mathbf{x}$ and $\mathbf{y}$ has the form

$$p_i(\mathbf{z}) = \frac{1}{(2\pi)^{\frac{N}{2}} \, |\mathbf{K}^{(i)}|^{\frac{1}{2}}} \exp\left[-\frac{1}{2} [\mathbf{z} - \mathbf{m}^{(i)}]^T [\mathbf{K}^{(i)}]^{-1} [\mathbf{z} - \mathbf{m}^{(i)}] \right] \quad i = 1,2 \tag{5}$$

where $\mathbf{z}$ is the observation vector with elements $\mathbf{x}$ and $\mathbf{y}$ and $\mathbf{m}^{(i)}$ is the mean vector

$$\mathbf{m}^{(i)} = \begin{bmatrix} \mathbf{m}_x^{(i)} \\ \mathbf{m}_y^{(i)} \end{bmatrix} \quad i = 1,2 \tag{6}$$

and $\mathbf{K}$ is the covariance matrix partitioned as follows

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_x^{(i)} & \mathbf{B}_{xy}^{(i)} \\ \mathbf{B}_{xy}^{(i)T} & \mathbf{K}_y^{(i)} \end{bmatrix} \quad i = 1,2 \tag{7}$$

Note that $\mathbf{K}_x^{(i)}$ is the covariance matrix for $\mathbf{x}$, $\mathbf{K}_y^{(i)}$ is the covariance matrix for $\mathbf{y}$, and $\mathbf{B}_{xy}^{(i)}$ is the cross covariance matrix between $\mathbf{x}$ and $\mathbf{y}$. The marginal and conditional densities are Gaussian [9] and are given by

$$p_i(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{N}{2}} \, |\mathbf{K}_x^{(i)}|^{\frac{1}{2}}} \exp\left[-\frac{1}{2} [\mathbf{x} - \mathbf{m}_x^{(i)}]^T [\mathbf{K}_x^{(i)}]^{-1} [\mathbf{x} - \mathbf{m}_x^{(i)}] \right] \quad i = 1,2 \tag{8}$$

$$p_i(\mathbf{y} \mid \mathbf{x}) = \frac{1}{(2\pi)^{\frac{N}{2}} \mid \mathbf{K}_{y\mid x}^{(i)} \mid^{\frac{1}{2}}} \exp\left[-\frac{1}{2} [\mathbf{y} - \mathbf{m}_{y\mid x}^{(i)}]^T [\mathbf{K}_{y\mid x}^{(i)}]^{-1} [\mathbf{y} - \mathbf{m}_{y\mid x}^{(i)}] \right]$$

$$i = 1,2 \qquad (9)$$

$$p_i(\mathbf{y}) = \frac{1}{(2\pi)^{\frac{N}{2}} \mid \mathbf{K}_y^{(i)} \mid^{\frac{1}{2}}} \exp\left[-\frac{1}{2} [\mathbf{y} - \mathbf{m}_y^{(i)}]^T [\mathbf{K}_y^{(i)}]^{-1} [\mathbf{y} - \mathbf{m}_y^{(i)}] \right]$$

$$i = 1,2 \qquad (10)$$

$$p_i(\mathbf{x} \mid \mathbf{y}) = \frac{1}{(2\pi)^{\frac{N}{2}} \mid \mathbf{K}_{x\mid y}^{(i)} \mid^{\frac{1}{2}}} \exp\left[-\frac{1}{2} [\mathbf{x} - \mathbf{m}_{x\mid y}^{(i)}]^T [\mathbf{K}_{x\mid y}^{(i)}]^{-1} [\mathbf{x} - \mathbf{m}_{x\mid y}^{(i)}] \right]$$

$$i = 1,2 \qquad (11)$$

where the conditional covariances and means have the form

$$\mathbf{K}_{y\mid x}^{(i)} = \mathbf{K}_y^{(i)} - \mathbf{B}_{xy}^{(i)T} [\mathbf{K}_x^{(i)}]^{-1} \mathbf{B}_{xy}^{(i)} , \quad i = 1,2 \qquad (12)$$

$$\mathbf{m}_{y\mid x}^{(i)} = \mathbf{m}_y^{(i)} + \mathbf{B}_{xy}^{(i)T} [\mathbf{K}_x^{(i)}]^{-1} [\mathbf{x} - \mathbf{m}_x^{(i)}] , \quad i = 1,2 \qquad (13)$$

Since the $\mathbf{x}$ term of (13) is not available at the given sensor, an estimate of the form

$$\hat{\mathbf{x}}_i = \mathbf{m}_x^{(i)} + \mathbf{B}_{xy}^{(i)} [\mathbf{K}_y^{(i)}]^{-1} [\mathbf{y} - \mathbf{m}_y^{(i)}] , \quad i = 1,2 \qquad (14)$$

is used. The estimate is the value of $\mathbf{x}$ that maximizes the density $p_i(\mathbf{x} \mid \mathbf{y})$. Symmetric forms of (12), (13), and (14) are used for $\mathbf{K}_{x\mid y}$, $\mathbf{m}_{x\mid y}$, and $\hat{\mathbf{y}}_i$ at the other sensor.

The natural logarithm of (8) is given by

$$\ln p_i(\mathbf{x}) = -\frac{1}{2}\left[\ln(2\pi)^N + \ln|\mathbf{K}_z^{(i)}| + [\mathbf{x} - \mathbf{m}_z^{(i)}]^T [\mathbf{K}_z^{(i)}]^{-1} [\mathbf{x} - \mathbf{m}_z^{(i)}]\right] \quad i = 1,2 \qquad (15)$$

and the natural logarithms of (9), (10), and (11) are similarly obtained. The logarithms of the conditional likelihood ratios are then used to obtain the terms on the right side of (1). The term given by

$$\lambda_A(\mathbf{x}_0) = \ln\frac{p_1(\mathbf{x}_0)}{p_2(\mathbf{x}_0)} = \ln p_1(\mathbf{x}_0) - \ln p_2(\mathbf{x}_0) \qquad (16)$$

then becomes

$$\lambda_A(\mathbf{x}_0) = -\frac{1}{2}\left[\ln|\mathbf{K}_z^{(1)}| + [\mathbf{x} - \mathbf{m}_z^{(1)}]^T [\mathbf{K}_z^{(1)}]^{-1} [\mathbf{x} - \mathbf{m}_z^{(1)}]\right]$$
$$+ \frac{1}{2}\left[\ln|\mathbf{K}_z^{(2)}| + [\mathbf{x} - \mathbf{m}_z^{(2)}]^T [\mathbf{K}_z^{(2)}]^{-1} [\mathbf{x} - \mathbf{m}_z^{(2)}].\right] \qquad (17)$$

Expanding (17) and collecting terms leads to the form

$$\lambda_A(\mathbf{x}_0) = \mathbf{x}^T \mathbf{A}\ \mathbf{x} + \mathbf{b}^T \mathbf{x} + c \qquad (18)$$

where

$$\mathbf{A} = \frac{1}{2}\left[[\mathbf{K}_z^{(2)}]^{-1} - [\mathbf{K}_z^{(1)}]^{-1}\right] \qquad (19)$$

is an NxN matrix,

$$\mathbf{b}^T = \left[[\mathbf{m}_z^{(1)}]^T [\mathbf{K}_z^{(1)}]^{-1} - [\mathbf{m}_z^{(2)}]^T [\mathbf{K}_z^{(2)}]^{-1}\right] \qquad (20)$$

is a 1xN vector, and

$$c = \frac{1}{2}\left[[\mathbf{m}_z^{(2)}]^T [\mathbf{K}_z^{(2)}]^{-1} \mathbf{m}_z^{(2)} - [\mathbf{m}_z^{(1)}]^T [\mathbf{K}_z^{(1)}]^{-1} \mathbf{m}_z^{(1)} + \ln\frac{|\mathbf{K}_z^{(2)}|}{|\mathbf{K}_z^{(1)}|}\right] \qquad (21)$$

is a scaler.

The coefficients of the conditional log likelihood ratio. $\lambda_A'(\mathbf{x}_0)$ are called $\mathbf{A}'$, $\mathbf{b}'$, and c' and are derived in the same way with (12), (13), and (14) substituted for the corresponding variables. Similar coefficients are calculated for $\lambda_B(\mathbf{y}_0)$ and

$\lambda_B'(\mathbf{y}_0)$ and are listed in Appendix A along with the coefficients for $\lambda_A(\mathbf{x}_0)$ and $\lambda_A'(\mathbf{x}_0)$. The computations of $\mathbf{A}$, $\mathbf{b}$, c, $\mathbf{A}'$, $\mathbf{b}'$, and c' are performed prior to their use in a real-time application and are input at the start of each process as the parameters for each of the quadratic classifiers.

III. THE TEST ENVIRONMENT

A. HARDWARE DESCRIPTION

The test environment for the distributed decision algorithms, designated Real-Time Cluster Star (RTC *), consists of a highly modular hardware base and a highly flexible operating system. The hardware consists of two clusters of single board computers (SBC's), each sharing a common backplane and an Ethernet local area network (LAN) serving as the communication link. Thus each cluster can be thought of as a node of a network and each node has multiple processors on a common bus.

1. The Cluster

The cluster configuration is diagramed in Figure 2. Each cluster consists of three SBC's physically connected by the MULTIBUS. Each SBC has 64K RAM of local memory and can access an additional 64K RAM board of shared memory and a 32K RAM board of common memory on the MULTIBUS. Also connected to the MULTIBUS are hard and floppy disk drives used for bootup and input/output operations.

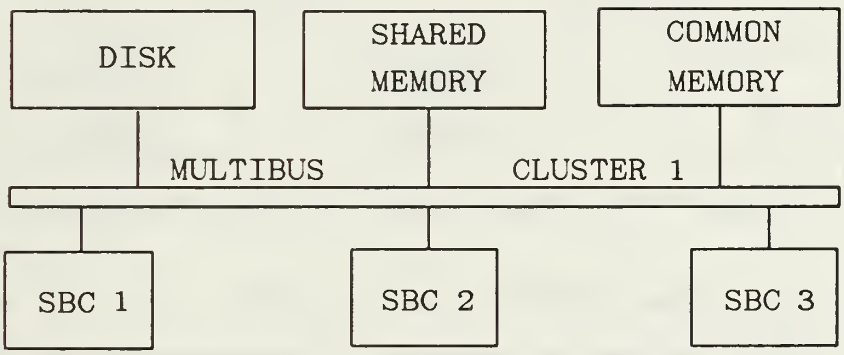

Figure 2 - Cluster Architecture

2. Real-Time Cluster Star (RTC *)

Figure 3 illustrates the RTC* architecture. It consists of two clusters connected by the Ethernet LAN. The Ethernet LAN/MULTIBUS interface is

the InterLAN NI3010 Ethernet Communications Controller Board (ECCB). This provides each cluster with its connection to the network. Further information on operating characteristics of the Ethernet LAN and RTC * use of the Ethernet LAN is available in [2,10].

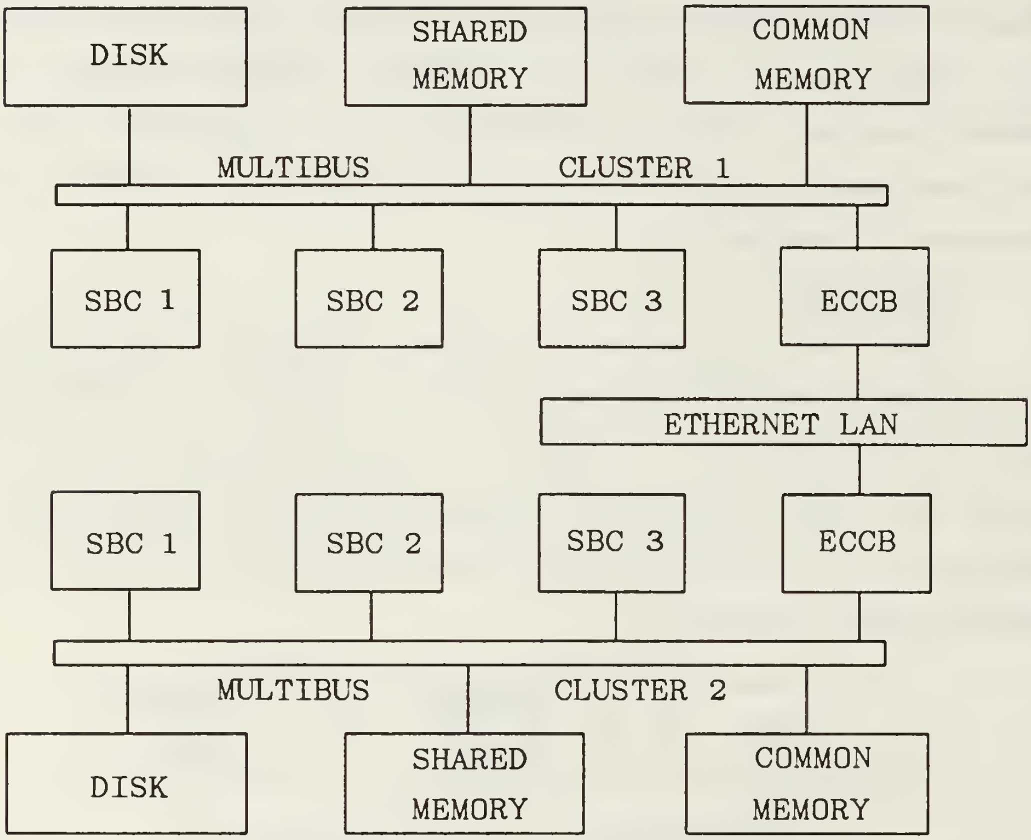

Figure 3 - Real-Time Cluster Star (RTC*) Architecture

B. THE OPERATING SYSTEM ENVIRONMENT

MCORTEX, the operating system, is a distributed multicomputer real-time executive. It allows for asynchronous operation of processes resident on SBC's in the same cluster and in separate clusters which are linked via the Ethernet LAN. System synchronization of computations in various distributed processes is accomplished using the synchronization model of Reed and Kanodia [11]. This section describes the MCORTEX system distribution of control variables, known

as **eventcounts** and **sequencers**. The modifications to the operating system, necessary to distribute user data throughout the system, are also discussed in this section.

1. The Synchronization Model

The MCORTEX operating system is based upon a synchronization model which is **event** oriented. Processes coordinate various activities by signaling and observing **events** using synchronization variables known as **eventcounts** and **sequencers**. An **eventcount** is a variable created by the user to signal the occurrence of an associated **event**. **Eventcounts** are initialized with the value zero and incremented by one each time the associated **event** occurs. The mechanism used to signal this occurrence is a call to a system primitive, the **advance**, which causes the **eventcount** to be incremented by one. A call to another system primitive, the **await**, causes a process to wait until the designated **eventcount** has reached a designated threshold. Once the **eventcount** value is equal to or greater than the threshold value the process may continue its execution. Therefore, processing at distributed locations may be controlled using **eventcounts** which are signaled and observed with the **advance** and the **await** primitives.

A **sequencer** is a variable provided by the system to control the allocation of a system shared resource. The **sequencer** is a positive integer number generator which starts with zero. It increments by one after providing its current value to any process which requests its associated shared resource. The **ticket** operation is the mechanism used to obtain a number from the **sequencer**. The number obtained is used as a threshold value in the **await** call to a system **eventcount** which is also associated with the shared resource. As users of the shared resource relinquish it, they increment the associated **eventcount** with the **advance**. This allows the user with the **ticket** value which matches the **eventcount** to gain access. An example of a shared resource controlled by a **sequencer** is the Ethernet LAN.

2. Eventcount Distribution

The kernel of MCORTEX is resident on each SBC and schedules processes for execution. A process runs until it invokes one of the system primitives, the **advance** or the **await**, which results in the actions described in Section B.1. The **advance** of an **eventcount**, which is used only within one cluster, causes an update of that clusters eventcount value. Processes in the same cluster, which are awaiting the eventcount, may then continue to execute. Update of eventcounts required for intercluster synchronization are **packetized** for transfer, via the Ethernet LAN, to the other cluster. The operating system procedure which accomplishes the transfer is located on SBC 1 of each cluster and is referred to in this thesis as the **driver**. The **driver** is the system software modified to allow for user data transfer between clusters.

3. Data Distribution

Data which must be shared between processes of the same cluster is made accessible through the use of pointers to access the local cluster **shared** memory locations. In the RTC* system, buffering of data must be done explicitly by user processes since no means of dynamic allocation presently exists. In this thesis, the real-time application requires the immediate use of the data generated, which precludes the need for buffering. Static storage locations, which are overwritten, are used for transfer of data throughout the system.

Data transfer from one cluster to another is accomplished by first establishing an absolute address in the local cluster **shared** memory to receive the data to be transferred. A pointer is used to access the absolute address in **shared** memory and the data value based at the pointer is updated. The system **driver** is then notified that a data value is ready for transfer. The Ethernet LAN sequencer provides the ticket to the user process for this data transfer. Once the ticket for this data value matches the eventcount associated with the Ethernet LAN, the data value is transferred to the driver's transmit data block in the appropriate data field in local cluster shared memory. The driver then causes the necessary calls to system subroutines to allow packetization and transfer over the Ethernet.

At the receiving end the message is processed by the local ECCB and the data is placed in the receive data block. The driver then stores the data value at the absolute address designated in the receiving clusters shared memory. Another pointer is then used in the receiving process to access the absolute address in shared memory. The data value based at the pointer is then available for further computations in this cluster. When the eventcount associated with this data transfer is updated via a similar procedure, the remaining computations are performed. User process eventcounts prevent the generation of additional data until the remaining computations in the present iteration are complete.

Appendix B provides an explanation of the steps necessary to create the system driver and user command files. The driver modifications required to transmit and receive data values for the distributed decision algorithms are shown in upper case lettering in the system procedure SYSDEV.PLI in Appendix C. User defined pointers and variable basing are shown and described further in the user procedures PA2, PA3, PB2, and PB3 in Appendix D.

C. ALGORITHM IMPLEMENTATION

Each cluster can be viewed as representing the set of local processors of a particular sensor which obtains large volumes of raw observation data from a target for initial processing. Decision rule parameters and raw observation data are read from local disk storage to the processes of two SBC's in a cluster. Two identical data sets are processed in parallel to generate a different reduced statistic in each processor. One statistic is to be used locally (at the same sensor) in further computation while the other is to be sent to the remote sensor for use in further computations. The local sensor then receives a reduced statistic from the remote sensor to combine with its locally retained statistic. The final result of the combined statistics is then compared to a decision threshold and the decision is displayed at a local sensor terminal.

1. Process Distributivity/Parallel Processing

The implementation of the decision rule described by (1) is accomplished with the following organization. The sensors associated with the two system

clusters, as well as the clusters themselves, are referred to as **SENSOR A** and **SENSOR B**. As illustrated in Figure 4, each sensor uses two processes labeled PA2(PB2) and PA3(PB3). Process computations take place in time order from left to right and computations shown above/below one another are performed in parallel.

SENSOR A

PA2: Compute $\lambda_A(X_0)$ Compute $\lambda_A(X_0) + \lambda_B'(Y_0)$

PA3: Compute $\lambda_A'(X_0)$

SENSOR B

PB3: Compute $\lambda_B'(Y_0)$

PB2: Compute $\lambda_B(Y_0)$ Compute $\lambda_B(Y_0) + \lambda_A'(X_0)$

Figure 4 - Computations of Reduced Statistics

Processes PA2(PB2) and PA3(PB3) are resident on SBC 2 and SBC 3, respectively, at each sensor. Computations are performed as shown, with the primed statistics exchanged between sensors to allow further computations in processes PA2 and PB2. The detailed computations discussed in Chapter II are shown in user processes PA2, PA3, PB2, and PB3 of Appendix D.

2. Process Synchronization

Synchronization of events during the decision rule computations is crucial for accurate and meaningful results. As illustrated in Figure 5, the careful synchronization of time critical events is coordinated with the use of two distributed eventcounts at each sensor. The A1EVC eventcount of Sensor A is advanced to signal the availability of the statistic $\lambda_A'(x_0)$ for use in PB2 of Sensor B and the B1EVC eventcount of Sensor B signals PA2 of Sensor A that $\lambda_B'(y_0)$ is available. The A2EVC and B2EVC eventcounts control the timing of the next input operation at both sensors to ensure correct correspondence of the

observation data. In distributed processing multicomputer systems, it is essential that all threshold values used in the calls to the await primitives for comparison to the eventcounts, be initialized properly to ensure continued operation of the real-time system.

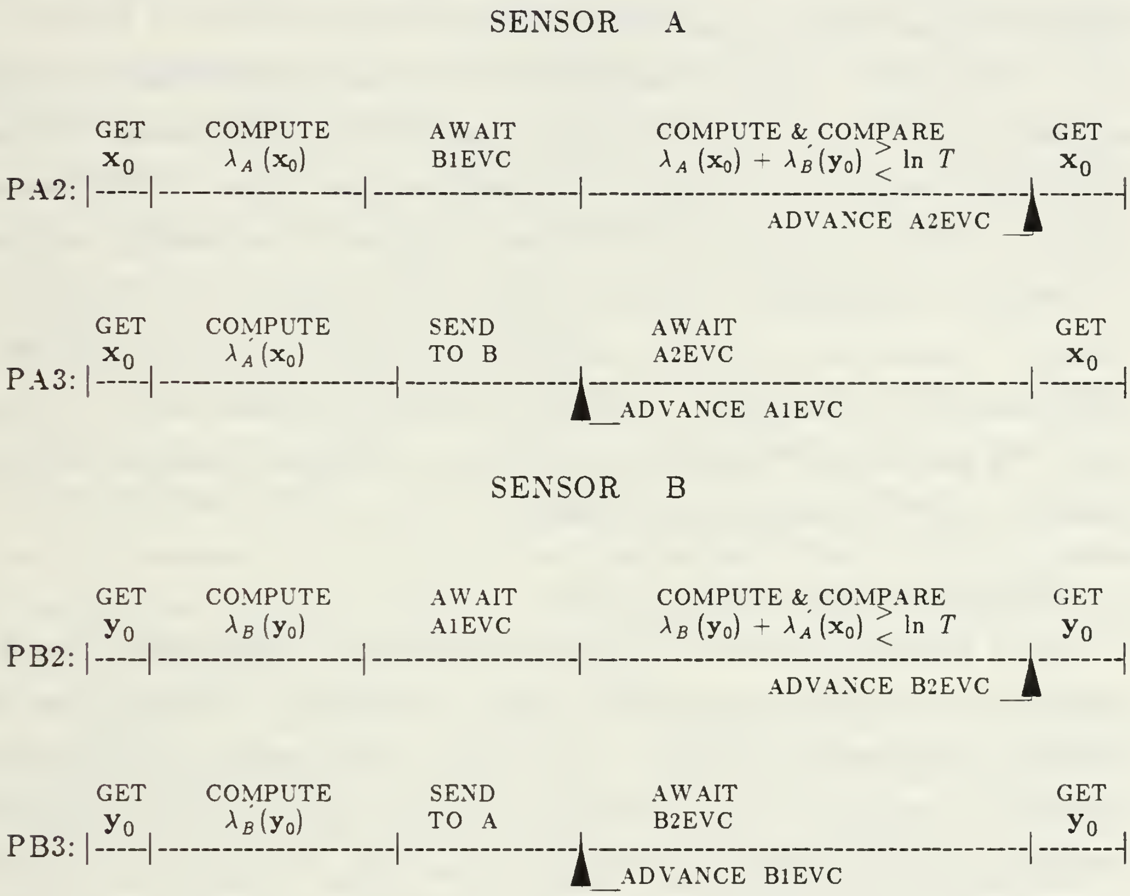

Figure 5 - Synchronization Diagram

As one might expect, there is a need to ensure that the required statistic, $\lambda_A'(x_0)$ or $\lambda_B'(y_0)$, is available for use prior to advancing the A1EVC or B1EVC eventcounts. This is insured by the forced synchronization of events inherent in the sensor to sensor transfer of user data and eventcount updates. The statistic to be transferred is stored in shared memory and transferred as described earlier. Once the Ethernet LAN sequencer ticket value is obtained for the data transfer and the request is placed in the ERB queue, the appropriate A1EVC or B1EVC

eventcount is advanced causing a system request for a ticket value from the same sequencer. This places the eventcount transfer request, which will signal the availability of data, behind the data transfer request in the same ERB queue. Therefore, when the eventcount is finally updated at the remote sensor the statistic required will be in place and available.

In the final stage of computation the reduced statistic retained locally and the statistic received from the remote sensor are added in processes PA2 and PB2 of each sensor and compared to a threshold (see Figure 2). The reduced statistics $\lambda_A(\mathbf{x}_0)$ and $\lambda_B^{'}(\mathbf{y}_0)$ are added and compared to the threshold at sensor A. Similarly, $\lambda_B(\mathbf{y}_0)$ and $\lambda_A^{'}(\mathbf{x}_0)$ are added and compared to the threshold at Sensor B. Results of the threshold decision are tabulated on the local consoles of each sensor and the loop begins again with the next observation vector read from disk. The processing of input observation vectors continues, simulating real-time operation until the vector files are depleted.

D. RESULTS OF THE SIMULATION

In the development and use of the test environment it was verified that it is important to distribute computation among processors to better utilize the available computational ability and minimize interprocess communication. Processes at each sensor were broken up and distributed among the available processors to gain increased computational advantages. Since processes at remote sensors had to be carefully synchronized, specific semaphore-like mechanisms were made available to provide this synchronization over the network. The specific mechanisms used in this implementation are the **await** and the **advance**. Correct operation of these synchronization mechanisms over the network depends on the prompt and orderly communication of protected variables used by the synchronization mechanisms. This orderly communication is achieved by the ticket operation. Successful implementation of a distributed decision algorithm requires the availability of all of these control mechanisms.

IV. CONCLUSIONS

The process of distributed decision making by two cooperating sensors observing a common phenomenon was introduced in this thesis. Decisions reached in this cooperative way produce more reliable results than those of sensors acting alone. Such decision procedures are characterized by the need to perform local computations at each sensor and to communicate partial results to the other sensor. Although several types of algorithms were cited to accomplish the desired distributed decision procedures, all have similar computation, communication, and process synchronization requirements.

A particular distributed decision algorithm based on the *generalized likelihood ratio test* was implemented to explore the computation, communication, and synchronization problems. The implementation was accomplished on a two node network connected via an Ethernet local area network. Each node of the network contained the required number of identical microprocessors sharing a common bus, shared memory, and network interfacing.

Problems of intercluster as well as intracluster synchronization of events between processes to ensure the timely input of observation data and the coordinated computation using the shared data from the opposite cluster were tested and resolved. Initial results using the *generalized likelihood ratio test* algorithm demonstrated the feasibility of performing the computations involved in the distributed decision algorithms in a realistic environment. The requirement for carefully designed, network-wide process control mechanisms was also found to be essential. The specific procedures used were discussed in the body of the thesis.

APPENDIX A

Quadratic Classifiers

Specific formulas for the quadratic classifiers, $\lambda_A(\mathbf{x}_0)$, $\lambda_A'(\mathbf{x}_0)$, $\lambda_B(\mathbf{y}_0)$, and $\lambda_B'(\mathbf{y}_0)$ described in Chapter II are provided in this appendix. Each quadratic classifier was derived similar to $\lambda_A(\mathbf{x}_0)$, in Chapter II, Section B. The coefficients, $\mathbf{A}$, $\mathbf{b}^T$, c, $\mathbf{A}'$, $\mathbf{b}^{T\prime}$, and c', the necessary expanding equations for variables $\mathbf{K}_{x|y}^{(i)}$, $\mathbf{m}_{x|y}^{(i)}$, $\mathbf{K}_{y|x}^{(i)}$, and $\mathbf{m}_{y|x}^{(i)}$, and the estimates, $\hat{\mathbf{y}}_i$ and $\hat{\mathbf{x}}_i$, are given as functions of the known terms, $\mathbf{K}_x^{(i)}$, $\mathbf{m}_x^{(i)}$, $\mathbf{K}_y^{(i)}$, $\mathbf{m}_y^{(i)}$, and $\mathbf{B}_{xy}^{(i)}$.

The coefficients computed for

$$\lambda_A(\mathbf{x}_0) = \mathbf{x}^T \mathbf{A}\ \mathbf{x} + \mathbf{b}^T \mathbf{x} + c$$

are

$$\mathbf{A} = \frac{1}{2}\left[[\mathbf{K}_x^{(2)}]^{-1} - [\mathbf{K}_x^{(1)}]^{-1} \right]$$

$$\mathbf{b}^T = \left[[\mathbf{m}_x^{(1)}]^T\ [\mathbf{K}_x^{(1)}]^{-1} - [\mathbf{m}_x^{(2)}]^T\ [\mathbf{K}_x^{(2)}]^{-1} \right]$$

$$c = \frac{1}{2}\left[[\mathbf{m}_x^{(2)}]^T\ [\mathbf{K}_x^{(2)}]^{-1}\ \mathbf{m}_x^{(2)} - [\mathbf{m}_x^{(1)}]^T\ [\mathbf{K}_x^{(1)}]^{-1}\ \mathbf{m}_x^{(1)} + \ln \frac{|\,\mathbf{K}_x^{(2)}\,|}{|\,\mathbf{K}_x^{(1)}\,|} \right]$$

The coefficients computed for

$$\lambda_A'(\mathbf{x}_0) = \mathbf{x}^T \mathbf{A}'\ \mathbf{x} + \mathbf{b}^{T\prime} \mathbf{x} + c'$$

are

$$\mathbf{A}' = \frac{1}{2}\left[[\mathbf{K}_{x|y}^{(2)}]^{-1} - [\mathbf{K}_{x|y}^{(1)}]^{-1} \right]$$

$$\mathbf{b}^{T\prime} = \left[[\mathbf{m}_{x|y}^{(1)}]^T\ [\mathbf{K}_{x|y}^{(1)}]^{-1} - [\mathbf{m}_{x|y}^{(2)}]^T\ [\mathbf{K}_{x|y}^{(2)}]^{-1} \right]$$

$$c' = \frac{1}{2}\left[[\mathbf{m}_{x|y}^{(2)}]^T \, [\mathbf{K}_{x|y}^{(2)}]^{-1} \, \mathbf{m}_{x|y}^{(2)} - [\mathbf{m}_{x|y}^{(1)}]^T \, [\mathbf{K}_{x|y}^{(1)}]^{-1} \, \mathbf{m}_{x|y}^{(1)} + \ln \frac{|\,\mathbf{K}_{x|y}^{(2)}\,|}{|\,\mathbf{K}_{x|y}^{(1)}\,|} \right]$$

$$\mathbf{K}_{x|y}^{(1)} = \mathbf{K}_x^{(1)} - \mathbf{B}_{xy}^{(1)} \, [\mathbf{K}_y^{(1)}]^{-1} \, \mathbf{B}_{xy}^{(1)T}$$

$$\mathbf{K}_{x|y}^{(2)} = \mathbf{K}_x^{(2)} - \mathbf{B}_{xy}^{(2)} \, [\mathbf{K}_y^{(2)}]^{-1} \, \mathbf{B}_{xy}^{(2)T}$$

$$\mathbf{m}_{x|y}^{(1)} = \mathbf{m}_x^{(1)} + \mathbf{B}_{xy}^{(1)} \, [\mathbf{K}_y^{(1)}]^{-1} \, [\mathbf{y} - \mathbf{m}_y^{(1)}]$$

$$\mathbf{m}_{x|y}^{(2)} = \mathbf{m}_x^{(2)} + \mathbf{B}_{xy}^{(2)} \, [\mathbf{K}_y^{(2)}]^{-1} \, [\mathbf{y} - \mathbf{m}_y^{(2)}]$$

$$\hat{\mathbf{y}}_1 = \mathbf{m}_y^{(1)} + \mathbf{B}_{xy}^{(1)T} \, [\mathbf{K}_x^{(1)}]^{-1} \, [\mathbf{x} - \mathbf{m}_x^{(1)}]$$

$$\hat{\mathbf{y}}_2 = \mathbf{m}_y^{(2)} + \mathbf{B}_{xy}^{(2)T} \, [\mathbf{K}_x^{(2)}]^{-1} \, [\mathbf{x} - \mathbf{m}_x^{(2)}]$$

The coefficients computed for

$$\lambda_B(\mathbf{y}_0) = \mathbf{y}^T \mathbf{A} \ \mathbf{y} + \mathbf{b}^T \mathbf{y} + c$$

are

$$\mathbf{A} = \frac{1}{2}\left[[\mathbf{K}_y^{(2)}]^{-1} - [\mathbf{K}_y^{(1)}]^{-1} \right]$$

$$\mathbf{b}^T = \left[[\mathbf{m}_y^{(1)}]^T \, [\mathbf{K}_y^{(1)}]^{-1} - [\mathbf{m}_y^{(2)}]^T \, [\mathbf{K}_y^{(2)}]^{-1} \right]$$

$$c = \frac{1}{2}\left[[\mathbf{m}_y^{(2)}]^T \, [\mathbf{K}_y^{(2)}]^{-1} \, \mathbf{m}_y^{(2)} - [\mathbf{m}_y^{(1)}]^T \, [\mathbf{K}_y^{(1)}]^{-1} \, \mathbf{m}_y^{(1)} + \ln \frac{|\,\mathbf{K}_y^{(2)}|}{|\,\mathbf{K}_y^{(1)}|} \right]$$

The coefficients computed for

$$\lambda_B'(\mathbf{y}_0) = \mathbf{y}^T \mathbf{A}' \ \mathbf{y} + \mathbf{b}^{T\prime} \mathbf{y} + c'$$

are

$$\mathbf{A}' = \frac{1}{2}\left[[\mathbf{K}_{y|x}^{(2)}]^{-1} - [\mathbf{K}_{y|x}^{(1)}]^{-1} \right]$$

$$\mathbf{b}^{T\,\prime} = \left[[\mathbf{m}_{y|x}^{(1)}]^T \; [\mathbf{K}_{y|x}^{(1)}]^{-1} - [\mathbf{m}_{y|x}^{(2)}]^T \; [\mathbf{K}_{y|x}^{(2)}]^{-1} \right]$$

$$c\,' = \frac{1}{2} \left[[\mathbf{m}_{y|x}^{(2)}]^T \; [\mathbf{K}_{y|x}^{(2)}]^{-1} \, \mathbf{m}_{y|x}^{(2)} - [\mathbf{m}_{y|x}^{(1)}]^T \; [\mathbf{K}_{y|x}^{(1)}]^{-1} \, \mathbf{m}_{y|x}^{(1)} + \ln \frac{|\, \mathbf{K}_{y|x}^{(2)} \,|}{|\, \mathbf{K}_{y|x}^{(1)} \,|} \right]$$

$$\mathbf{K}_{y|x}^{(1)} = \mathbf{K}_y^{(1)} - \mathbf{B}_{xy}^{(1)T} \, [\mathbf{K}_x^{(1)}]^{-1} \, \mathbf{B}_{xy}^{(1)}$$

$$\mathbf{K}_{y|x}^{(2)} = \mathbf{K}_y^{(2)} - \mathbf{B}_{xy}^{(2)T} \, [\mathbf{K}_x^{(2)}]^{-1} \, \mathbf{B}_{xy}^{(2)}$$

$$\mathbf{m}_{y|x}^{(1)} = \mathbf{m}_y^{(1)} + \mathbf{B}_{xy}^{(1)T} \, [\mathbf{K}_x^{(1)}]^{-1} \, [\mathbf{x} - \mathbf{m}_x^{(1)}]$$

$$\mathbf{m}_{y|x}^{(2)} = \mathbf{m}_y^{(2)} + \mathbf{B}_{xy}^{(2)T} \, [\mathbf{K}_x^{(2)}]^{-1} \, [\mathbf{x} - \mathbf{m}_x^{(2)}]$$

$$\hat{\mathbf{x}}_1 = \mathbf{m}_x^{(1)} + \mathbf{B}_{xy}^{(1)} \, [\mathbf{K}_y^{(1)}]^{-1} \, [\mathbf{y} - \mathbf{m}_y^{(1)}]$$

$$\hat{\mathbf{x}}_2 = \mathbf{m}_x^{(2)} + \mathbf{B}_{xy}^{(2)} \, [\mathbf{K}_y^{(2)}]^{-1} \, [\mathbf{y} - \mathbf{m}_y^{(2)}]$$

APPENDIX B
LINK86 Input Option Files

When linking files to create a command file for use on each SBC, the following command is invoked with the appropriate user filename: LINK86 filename [I]. The "I" in square brackets invokes the input file option which directs LINK86 to obtain further command line input from the designated input file. As an example, the modules listed in CA.INP are linked with the command: LINK86 CA[I], where the "I" indicates that CA.INP contains the names of the files to be linked. The name preceding the equal sign is the filename assigned to the command file. LINK86 CA[I] produces the command file CA.CMD, which is the system driver for Sensor A (cluster A). All files listed in the input file must be on the logon disk and must be of type object (.obj). Object files are generated by compiling files of type PLI (.pli) or A86 (.a86). The above steps also apply for linking the system driver files for Sensor B as well as the user files, processes PA2, PA3, PB2, and PB3, to create the respective command files CB.CMD, NUM12.CMD, NUM13.CMD, NUM22.CMD, and NUM23.CMD.

```
************************************************************
************************************************************
**                CA.INP input option file                **
************************************************************
************************************************************

CA =
SYSINITA [CODE[AB[439]],DATA[AB[800],M[Ø],AD[82]],MAP[ALL]],
SYSDEV,
ASMROUT,
GATEMOD

************************************************************
************************************************************
**               NUM12.INP input option file              **
************************************************************
************************************************************

NUM12=
SBC2INIT [CODE[AB[439]],DATA[AB[800],M[Ø],AD[82]],MAP[ALL]],
PA2,
GATEMOD

************************************************************
************************************************************
**               NUM13.INP input option file              **
************************************************************
************************************************************

NUM13=
SBC3INIT [CODE[AB[439]],DATA[AB[800],M[Ø],AD[82]],MAP[ALL]],
PA3,
GATEMOD
```

```
****************************************************************
****************************************************************
**                  CB.INP input option file                  **
****************************************************************
****************************************************************

CB =
SYSINITB [CODE[AB[439]],DATA[AB[800],M[0],AD[82]],MAP[ALL]],
SYSDEV,
ASMROUT,
GATEMOD

****************************************************************
****************************************************************
**                NUM22.INP input option file                 **
****************************************************************
****************************************************************

NUM22=
SEC2INIT [CODE[AB[439]],DATA[AB[800],M[0],AD[82]],MAP[ALL]],
PB2,
GATEMOD

****************************************************************
****************************************************************
**                NUM23.INP input option file                 **
****************************************************************
****************************************************************

NUM23=
SEC3INIT [CODE[AB[439]],DATA[AB[800],M[0],AD[82]],MAP[ALL]],
PB3,
GATEMOD
```

APPENDIX C
Device Driver and Packet Processor Source Code

This code consists of PL/I-86 and 8086 assembly language modules. When linked as described in Appendix A and loaded in local memory of SBC #1 of each cluster, the driver handles the systemwide distribution of user data and eventcounts via the local area network.

Initialization modules (SYSINITA & SYSINITB), each for their own cluster, define cluster addresses, create user eventcounts, establish eventcount distribution, and create the procedure space, under operating system control, for the driver, SYSDEV. The system definitions file, SYSDEF and the file NI3010.DCL are required when compiling SYSDEV. Any user eventcounts, sequencers, or shared variable pointers which are defined in SYSDEF must be updated when these items change with new synchronization and control schemes.

SYSINITA and SYSINITB must also be updated whenever changes are made to user eventcounts or their distribution. Recompilation and relinking are also necessary to produce the updated command files CA.CMD and CB.CMD.

```
*************************************************************
*************************************************************
**                                                         **
**   CLUSTER A  INITIALIZATION MODULE  SYSINITA.PLI        **
**                                                         **
*************************************************************
*************************************************************

SYSINITA: proc options (main);

     %include 'sysdef.pli';

     %replace

   EVC_TYPE         by '00'b4;

     /* main */

   call define_cluster ('0001'b4); /* must be called
                                      prior to creating
                                      evc's  */

   /**** USER ****/

   CALL CREATE_EVC (A1EVC);

   CALL CREATE_EVC (A2EVC);

        CALL CREATE_EVC (B1EVC);

   /*** SYSTEM ***/


   call create_evc (ERB_READ);
   call create_evc (ERB_WRITE);
   call create_seq (ERB_WRITE_REQUEST);

   /* distrib. map called after eventcounts have
      been created */

   /* local and remote copy of  A1EVC needed */

   call distribution_map (EVC_TYPE, A1EVC, '0003'B4);

   call create_proc ('fc'b4, '80'b4,
                     '0941'b4, '0800'b4, '0053'b4,
                     '0439'b4, '0800'b4, '0800'b4);

   call await ('fe'b4, '01'b4);

END SYSINITA;
```

```
*******************************************************************
*******************************************************************
**                                                             **
**   CLUSTER B  INITIALIZATION MODULE  SYSINITB.PLI            **
**                                                             **
*******************************************************************
*******************************************************************

SYSINITB: proc options (main);

     %include 'sysdef.pli';

     %replace

   EVC_TYPE          by '00'b4;

     /* main */

   call define_cluster ('0002'b4); /* must be called
                                      prior to creating
                                      evc's  */

   /**** USER ****/

   CALL CREATE_EVC (A1EVC);

   CALL CREATE_EVC (B1EVC);

        CALL CREATE_EVC (B2EVC);

   /*** SYSTEM ***/

   call create_evc (ERB_READ);
   call create_evc (ERB_WRITE);

   call create_seq (ERB_WRITE_REQUEST);

        /* distrib. map called after eventcounts have
    been created */

   /* local and remote copy of  B1EVC needed */

   call distribution_map (EVC_TYPE, B1EVC, '0003'B4);

   call create_proc ('fc'b4, '80'b4,
                     '0941'b4, '0800'b4, '0053'b4,
                     '0439'b4, '0800'b4, '0800'b4);

   call await ('fe'b4, '01'b4);

END SYSINITB;
```

```
/*****************************************************************/
/*****************************************************************/
/**     FILE SYSDEF.PLI      MARK A. SCHON     24 JUL 85       **/
/**===========================================================**/
/** This section of code is given as a PLI file to be          **/
/** %INCLUDE'd with SYSDEV.PLI. ENTRY declarations are         **/
/** made for all available MCORTEX functions.                  **/
/*****************************************************************/
/*****************************************************************/

    DECLARE

      advance ENTRY (BIT (8)),
        /* advance (event_count_id) */
      await ENTRY (BIT (8), BIT (16)),
        /* await (event_count_id, awaited_value) */
      create_evc ENTRY (BIT (8)),
        /* create_evc (event_count_id) */

      create_proc ENTRY (BIT (8), BIT (8),
                         BIT (16), BIT (16), BIT (16),
                         BIT (16), BIT (16), BIT (16)),
        /* create_proc (processor_id, processor_priority, */
        /*              stack_pointer_highest, stack_seg, ip */
        /*              code_seg, data_seg, extra_seg)      */

      create_seq ENTRY (BIT (8)),
        /* create_seq (sequence_id) */

      preempt ENTRY (BIT (8)),
        /* preempt (processor_id) */

      read ENTRY (BIT (8)) RETURNS (BIT (16)),
        /* read (event_count_id) */
        /* RETURNS current_event_count */

      ticket ENTRY (BIT (8)) RETURNS (BIT (16)),
        /* ticket (sequence_id) */
        /* RETURNS unique_ticket_value */

      define_cluster ENTRY  (bit (16)),

        /* define_cluster (local_cluster_address) */

      distribution_map ENTRY (bit (8), bit (8), bit (16)),

/* distribution_map (distribution_type, id, cluster_addr) */

      add2bit16 ENTRY (BIT(16), BIT(16)) RETURNS (BIT (16));
        /* add2bit16 ( a_16bit_#, another_16bit_#) */
        /* RETURNS a_16bit_# + another_16bit_#     */
```

```
%replace
  /*-----------------------------------------------------

        ***  EVC$ID's ***

     (1) USER                                            */

     A1EVC                        BY '01'B4,

     A2EVC                        BY '02'B4,

B1EVC                        BY '03'B4,

     B2EVC                        BY '04'B4,

  /* (2) SYSTEM                                          */

ERB_READ                     by 'fc'b4,
ERB_WRITE                    by 'fd'b4,

/*------------------------------------------------------

       ***    SEQUENCER NAMES    ***

          (1) USER

          USER PROCESSES USE ERB_WRITE_REQUEST ONLY.

     (2) SYSTEM  */

ERB_WRITE_REQUEST            by 'ff'b4,

/*------------------------------------------------------

    ***  SHARED VARIABLE POINTERS   ***

     (1) USER                                          */

     PB                           BY '8CC0'B4,
     PC                           BY '8DD0'B4,

/*           (2) SYSTEM  */

block_ptr_value              by '8000'b4,

xmit_ptr_value               by '80C8'b4,

rcv_ptr_value                by '8666'b4,

END_RESERVE                  by 'FFFF'b4;
```

```
***********************************************************
***********************************************************
**                                                       **
**                   NI3010.DCL FILE                     **
**                                                       **
***********************************************************
***********************************************************

     %replace

/*                   I/O port addresses

     These values are specific to the  use of the INTERLAN
 NI3010 MULTIBUS to ETHERNET interface  board.  Any change
 to the I/O port address of '00b0' hex (done so with a DIP
switch) will require a change to these addresses to  reflect
that change.                                          */

               command_register               by 'b0'b4,
               command_status_register        by 'b1'b4,
               transmit_data_register         by 'b2'b4,
               interrupt_status_reg           by 'b5'b4,
               interrupt_enable_register      by 'b8'b4,
               high_byte_count_reg            by 'bc'b4,
               low_byte_count_reg             by 'bd'b4,

          /*  end of I/O port addresses */

          /*  Interrupt enable status register values */
               disable_ni3010_interrupts      by '00'b4,
               ni3010_intrpts_disabled        by '00'b4,
               receive_block_available        by '04'b4,
               transmit_dma_done              by '06'b4,
               receive_dma_done               by '07'b4,

          /*   end register values */

          /*   Command Function Codes */

               module_interface_loopback      by '01'b4,
               internal_loopback              by '02'b4,
               clear_loopback                 by '03'b4,
          go_offline                      by '08'b4,
          go_online                       by '09'b4,
               onboard_diagnostic              by '0a'b4,
          clr_insert_source               by '0e'b4,
          load_transmit_data              by '28'b4,
          load_and_send                   by '29'b4,
          load_group_addresses            by '2a'b4,
          reset                           by '3f'b4;
          /*   end Command Function Codes        */
```

```
*************************************************************
*************************************************************
**                                                         **
**   CLUSTER A  - ADDRESS.DAT FILE - USED BY SYSDEV.PLI    **
**              -  1ST THREE VALUES USED IN SUBROUTINE     **
**                 program_group_addresses                 **
**              -  LAST TWO USED IN MAIN PROGRAM SYSDEV    **
**                 TO IDENTIFY THE LOCAL CLUSTER ADDRESS.  **
**                                                         **
*************************************************************
*************************************************************

1,
'00000000'b,'00000001'b,
'00000000'b,'00000001'b

*************************************************************
*************************************************************
**                                                         **
**   CLUSTER B  - ADDRESS.DAT FILE - USED BY SYSDEV.PLI    **
**              -  1ST THREE VALUES USED IN SUBROUTINE     **
**                 program_group_addresses                 **
**              -  LAST TWO USED IN MAIN PROGRAM SYSDEV    **
**                 TO IDENTIFY THE LOCAL CLUSTER ADDRESS.  **
**                                                         **
*************************************************************
*************************************************************

1,
'00000000'b,'00000010'b,
'00000000'b,'00000010'b


SYSDEV: PROCEDURE;

/*   Date:                 24 JULY 1985

     Programmer:           MARK A. SCHON (MODIFIED CODE FROM
                                          PREVIOUS THESIS[2])

     Module Function: To serve as the Ethernet Communication
                Controller Board, ECCB (NI3010) device
                handler.  This process is scheduled
                under MCORTEX and consumes Ethernet
                Requests Packets (ERP) generated by
                the  SYSTEM$IO located in LEVEL2.SRC &
                     by USER PROGRAMS.
                    It also processes any  inbound
                     packets  by  analyzing the packet
                     contents and making the appropriate
                     MCORTEX calls.   */
```

```
%replace

   evc_type              by '00'b4,
   erb_block_len         by 20,
   erb_block_len_m1      by 19,
   infinity              by 32767;

%include 'sysdef.pli';


DECLARE


1     erb(0:erb_block_len_m1) based (block_ptr),
            2 command      bit (8),
            2 type_name    bit (8),
            2 name_value   bit (16),
            2 remote_addr  bit (16),


            1     transmit_data_block based (xmit_ptr),

            2 destination_address_a
                 bit (8) ,
            2 destination_address_b
                 bit (8),
            2 destination_address_c
                 bit (8),
                 2 destination_address_d
                 bit (8),
            2 destination_address_e
                 bit (8) ,
            2 destination_address_f
                 bit (8) ,
            2 source_address_a
                 bit (8) ,
            2 source_address_b
                 bit (8),
            2 source_address_c
                 bit (8),
                 2 source_address_d
                 bit (8),
            2 source_address_e
                 bit (8) ,
            2 source_address_f
                 bit (8) ,
                 2 type_field_a
                      bit (8) ,
                 2 type_field_b
                      bit (8),
                 2 data (4) bit (8),
```

```
                 2 USER_DATA (12) FLOAT,

               (TX_DATA_PTR,XMIT_PTR) POINTER,

/* HIGH MEMORY ADDRESSES OF TX_DATA_PTR AND XMIT_PTR          */
/*                              ASSIGNED IN SYSDEV            */

               DATA_TO_SEND FLOAT BASED(TX_DATA_PTR),

               1 receive_data_block based (rcv_ptr),

                    2 frame_status               bit (8),
                    2 null_byte                  bit (8) ,
                    2 frame_length_lsb           bit (8) ,
                    2 frame_length_msb           bit (8) ,
                    2 destination_address_a      bit (8) ,
                    2 destination_address_b      bit (8) ,
                    2 destination_address_c      bit (8) ,
                    2 destination_address_d      bit (8) ,
                    2 destination_address_e      bit (8) ,
                    2 destination_address_f      bit (8) ,
                    2 source_address_a           bit (8) ,
                    2 source_address_b           bit (8) ,
                    2 source_address_c           bit (8) ,
                    2 source_address_d           bit (8) ,
                    2 source_address_e           bit (8) ,
                    2 source_address_f           bit (8) ,
                    2 type_field_a               bit (8) ,
                    2 type_field_b               bit (8) ,
                    2 data(4)                    bit (8) ,
                    2 USER_DATA (12)             FLOAT,
                    2 crc_msb                    bit (8) ,
                    2 crc_upper_middle_byte      bit (8) ,
                    2 crc_lower_middle_byte      bit (8) ,
                    2 crc_lsb                    bit (8) ,

               (RX_DATA_PTR,RCV_PTR,BLOCK_PTR) POINTER,

/* HIGH MEMORY ADDRESSES OF RX_DATA_PTR,RCV_PTR,&             */
/*          BLOCK_PTR ARE ASSIGNED IN FILE SYSDEF             */

               DATA_ARRIVED FLOAT BASED(RX_DATA_PTR),

            index fixed bin (15),
            (addr_e, addr_f) bit (8),
            address file,
                 copy_ie_register bit (8),
          (cluster_addr,erb_write_value,i) bit (16).
                 (j,k) fixed bin (15),
            reg_value bit (8) ,
            write_io_port entry (bit (8), bit (8)),
```

```
                    read_io_port  entry (bit (8), bit (8)),

                    initialize_cpu_interrupts     entry,
                    enable_cpu_interrupts         entry,
                    disable_cpu_interrupts        entry,
                    write_bar entry (bit(16));

                    /*   end module listing  */

        %replace

    /* codes specific to the Intel 8259a Programmable
                           Interrupt Controller (PIC)      */

                      icw1_port_address          by 'c2'b4,
                      icw2_port_address          by 'c2'b4,
                      icw4_port_address          by 'c2'b4,
                      ocw_port_address           by 'c2'b4,

              /* note: icw ==> initialization control word

                      ocw ==> operational command word   */

                      icw1                       by '13'b4,

          /* single PIC configuration, edge triggered input */

                      icw2                       by '40'b4,

          /* most significant bits of vectoring byte; for an
             interrupt 5,the effective address will be
             (icw2 + interrupt #) * 4 which will be
             (40 hex + 5) * 4 =114 hex                     */

                      icw4                       by '0f'b4,

  /* automatic end of interrupt and buffered mode/master   */

                      ocw1                       by '3f'b4;


                 /* unmask interrupt 4 (bit 4),            */
                    /* interrupt 5 (bit 5), and
 */
                 /* interrupt 6 (bit 6), mask all others */


                            /* end 8259a codes */

 /* include constants specific to the NI3010 board         */
                    %include 'ni3010.dcl';
```

```
/*************************************************************************/

                        /*  Main Body */

  call write_io_port(interrupt_enable_register,
               disable_ni3010_interrupts);
  call initialize_pic;
  call initialize_cpu_interrupts;
  call read_io_port (command_status_register,reg_value);
  call perform_command (reset);

  call program_group_addresses;
  /* assignments to the source and destination address
     fields that will not change */

  call perform_command (clr_insert_source);
  /* NI3010 performance is enhanced in this mode */

/**********************************************************************/
/* ASSIGN POINTER VALUES, PREVIOUSLY DEFINED -FILE SYSDEF */
/*     TX_DATA_PTR <-- PB = 8CC0 A FLOAT BLOCK OF 4-BYTES */
/*     RX_DATA_PTR <-- PC = 8DD0 A FLOAT BLOCK OF 4-BYTES */
/*                                                        */
/* BLOCK_PTR <-- BLOCK_PTR_VALUE = 8000 THE ECB 120-BYTES */
/* RCV_PTR   <-- RCV_PTR_VALUE   = 8666 THE RDB  66-BYTES */
/* XMIT_PTR  <-- XMIT_PTR_VALUE  = 80C8 THE TDB  74-BYTES */
/**********************************************************************/

        UNSPEC(TX_DATA_PTR) = PB;
        UNSPEC(RX_DATA_PTR) = PC;
     unspec(block_ptr) = block_ptr_value;
     unspec(rcv_ptr) =  rcv_ptr_value;
     unspec(xmit_ptr) = xmit_ptr_value;

  /* make one time assignments to transmit data block */

  transmit_data_block.destination_address_a = '03'b4;
  transmit_data_block.destination_address_b = '00'b4;
  transmit_data_block.destination_address_c = '00'b4;
  transmit_data_block.destination_address_d = '00'b4;
  transmit_data_block.source_address_a = '03'b4;
  transmit_data_block.source_address_b = '00'b4;
  transmit_data_block.source_address_c = '00'b4;
  transmit_data_block.source_address_d = '00'b4;

  /* get the local cluster address - file was
     opened in proc program_group_addresses   */

  get file (address) list (addr_e, addr_f);
  transmit_data_block.source_address_e = addr_e;
  transmit_data_block.source_address_f = addr_f;
```

```
cluster_addr = addr_e || addr_f;
put skip (2) edit ('*** CLUSTER ',cluster_addr,
               ' Initialization Complete ***')
               (col(15),a,b4(4),a);
i = '0001'b4;
call perform_command (go_online);

/* at this point copy_ie_reg = RBA , but
   ie_reg on NI3010 is actually disabled */
call disable_cpu_interrupts;

do k = 1 to infinity;
   /* note: interrupt not allowed during a
       call to MCORTEX primitive       */

   erb_write_value = read(ERB_WRITE);
 /* In the MXTRACE version of the RTOS
    all primitive calls clear    and
    set interrupts (diagnostic message
    routines), so the NI3010 interrupts
    must be disabled on entry to MXTRACE */
   do while (erb_write_value < i);
 /* busy waiting */
      erb_write_value = read(ERB_WRITE);
 copy_ie_register=receive_block_available;
 call write_io_port(interrupt_enable_register,
                 receive_block_available);
      call enable_cpu_interrupts;
   /* if a packet has been received,this
      is when an interrupt may occur - can
      see that outbound packets are always
      favored.   */
 do j = 1 to 1000;
  /* interrupt window for packets received */
 end; /* do j */
 call disable_cpu_interrupts;
 if (copy_ie_register = receive_dma_done) then
 do;
   /* receive DMA operation started, so let
      finish. */
   call enable_cpu_interrupts;
   do while (copy_ie_register = receive_dma_done);
   end;
   call disable_cpu_interrupts;
 end; /* ift */

      copy_ie_register = disable_ni3010_interrupts;
      call write_io_port(interrupt_enable_register,
                         disable_ni3010_interrupts);

   end; /* busy */
```

```
    /* ERB has an ERP in it, so process it */
    /* no external interrupts (RBA) until
       the ERP is consumed and the packet
       gets sent    */
      index = mod((fixed(i) - 1), erb_block_len);
          /* 32k limit on parameter to fixed fcn. */

      transmit_data_block.data(1) = erb(index).command;
      transmit_data_block.data(2) = erb(index).type_name;

      transmit_data_block.data(3) =
                          substr(erb(index).name_value,9.8);

      transmit_data_block.data(4) =
                          substr(erb(index).name_value,1.8);

      IF (ERB(INDEX).COMMAND = 1) THEN DO;

            TRANSMIT_DATA_BLOCK.USER_DATA(1) = DATA_TO_SEND;

      END;

      transmit_data_block.destination_address_e=
                        substr(erb(index).remote_addr, 1,8);

      transmit_data_block.destination_address_f=
                        substr(erb(index).remote_addr, 9,8);

      call advance (ERB_READ); /* caution here !!!'
                     an ADVANCE will result in a
                     call to VP$SCHEDULER, which
                     will set CPU interrupts on exit.
                     It's the reason NI3010 interrupts
                     are disabled first in the
                     Do While loop above.  */

    /* packet ready to go, so send it */

    call transmit_packet;

    /* copy_ie_register = RBA , but not actual register */

    call disable_cpu_interrupts;

    /* setting up for next ERP consumption */

    i = add2bit16(i, '0001'b4);

 end;  /* do forever */

    /* end main body */
```

```
/**********************************************************************/


initialize_pic:      procedure;

        DECLARE

             write_io_port entry (bit (8) , bit(8));

   call write_io_port (icw1_port_address,icw1);
   call write_io_port (icw2_port_address,icw2);
   call write_io_port (icw4_port_address,icw4);
   call write_io_port (ocw_port_address,ocw1);

end initialize_pic;

/**********************************************************************/

perform_command:   procedure (command);

   DECLARE
           command bit (8) ,
           reg_value bit (8) ,
           srf bit (8) ,
           write_io_port entry (bit (8) ,bit (8) ),
           read_io_port  entry (bit (8) ,bit (8) );

   /* end declarations */

   srf = '0'b4;
   call write_io_port (command_register,command);
   do while (¬(srf & '01'b4) = '00'b4);
      call read_io_port (interrupt_status_reg,srf);
   end;  /* do while */
        call read_io_port
        (command_status_register,reg_value);
   if (reg_value > '01'b4) then
   do;
      /* not (SUCCESS or SUCCESS with Retries) */

      put skip edit ('*** ETHERNET  Board Failure ***')
                     (col(20),a);
              /* when this occurs, run the diagnostic
                 routine T3010/Cx, where x is the
                 current cluster number */
           stop;
   end; /* itd */

end perform_command;
```

```
/*******************************************************************/

transmit_packet: procedure  external;


  DECLARE
     srf bit (8) ,
     reg_value bit (8) ,
     write_io_port entry (bit (8) ,bit (8) ),
     read_io_port entry  (bit (8) ,bit (8) ),
     enable_cpu_interrupts          entry,
     disable_cpu_interrupts         entry,
     write_bar entry (bit(16));

     /* begin */

    srf = '0'b4;

  call write_bar (xmit_ptr_value);
  call write_io_port(high_byte_count_reg,'00'b4);
  call write_io_port(low_byte_count_reg,'3c'b4);
  copy_ie_register = transmit_dma_done;
  call write_io_port(interrupt_enable_register,
                                      transmit_dma_done);
  call enable_cpu_interrupts;
  do while (copy_ie_register = transmit_dma_done);
  end;    /* loop until the interrupt handler
             takes care of the TDD interrupt -
             it sets copy_ie_register = RBA */
  call perform_command (load_and_send);

end transmit_packet;

/*****************************************************************
**********/


HL_interrupt_handler: procedure external;


   /* This routine is called from the low level
      8086 assembly language interrupt routine */

   DECLARE

      write_io_port entry (bit (8) ,bit (8) ),
      read_io_port entry (bit (8) ,bit (8) ),
      enable_cpu_interrupts          entry,
      disable_cpu_interrupts         entry,
      write_bar entry (bit(16));
```

```
   /*  begin   */

   call write_io_port(interrupt_enable_register,
                      disable_ni3010_interrupts);

   if (copy_ie_register = receive_block_available)
   then do;

      call write_bar (rcv_ptr_value);
      call write_io_port(high_byte_count_reg,'05'b4);
      call write_io_port(low_byte_count_reg,'f2'b4);

      /* initiate receive DMA */

      copy_ie_register = receive_dma_done;
      call write_io_port(interrupt_enable_register,
                                    receive_dma_done);

   end;   /* do */
   else
      if (copy_ie_register = receive_dma_done) then
      do;
         call process_packet;
         copy_ie_register = receive_block_available;
         call write_io_port(interrupt_enable_register,
                            receive_block_available);
      end;  /* if then do */
      else
         if (copy_ie_register = transmit_dma_done)
         then do;

         copy_ie_register = receive_block_available;
         /* NI3010 interrupts disabled on entry */
      end;   /* if then do */

end HL_interrupt_handler;


/****************************************************************/

process_packet: procedure;

   DECLARE

      DATA_ARRIVED FLOAT BASED(RX_DATA_PTR),

      RX_DATA_PTR POINTER,

      local_evc_value bit (16),
      data_ptr pointer,
      remote_evc_value bit (16) based (data_ptr);
```

```
      if (receive_data_block.data(1) = evc_type) then
      do;
         data_ptr = addr(receive_data_block.data(3));

         /* remote_evc_value now has a value */

         local_evc_value = read(receive_data_block.data(2));

         do while (local_evc_value < remote_evc_value);

       call advance (receive_data_block.data(2));
       local_evc_value = add2bit16(local_evc_value,
                                               '0001'b4);

         end;
    call disable_cpu_interrupts;
    /* this must be done due to setting of
       cpu interrupts by calls to MCORTEX's
       VP$SCHEDULER via ADVANCE */

      end; /* itd */

/*  IF DATA IS IN THIS RDB THEN TRANSFER IT TO USER HIGH   */
/*                                                 MEMORY  */

      ELSE DO;

         UNSPEC(RX_DATA_PTR) = PC;
         DATA_ARRIVED = RECEIVE_DATA_BLOCK.USER_DATA(1);

      END;

end process_packet;

/*************************************************************************/

program_group_addresses: procedure;

   DECLARE


   1  group_addr(40) based (group_ptr),

      2 mc_group_field_a
           bit (8),
      2 mc_group_field_b
           bit (8),
      2 mc_group_field_c
           bit (8),
      2 mc_group_field_d
           bit (8),
```

```
      2 mc_group_field_e
            bit (8),
      2 mc_group_field_f
            bit (8);


   DECLARE

   (group_ptr,p) pointer,
   (field_e, field_f) bit (8),
   bit_8_groups bit (8) based (p),
   (i,num_groups,groups_times_6) fixed bin (7);

    unspec(group_ptr) = xmit_ptr_value;
    open file (address) stream input;
    get file (address) list (num_groups);
    do i = 1 to num_groups;

       group_addr(i).mc_group_field_a = '03'b4;
       group_addr(i).mc_group_field_b = '00'b4;
       group_addr(i).mc_group_field_c = '00'b4;
       group_addr(i).mc_group_field_d = '00'b4;
       get file (address) list (field_e,field_f);
       group_addr(i).mc_group_field_e = field_e;
       group_addr(i).mc_group_field_f = field_f;

    end;    /* do i */

    call disable_cpu_interrupts;
    call write_bar (xmit_ptr_value);
    call write_io_port(high_byte_count_reg, '00'b4);
    groups_times_6 = 6 * num_groups;
    p = addr (groups_times_6);
    call write_io_port(low_byte_count_reg, bit_8_groups);
    copy_ie_register = transmit_dma_done;
    call write_io_port(interrupt_enable_register,
                       transmit_dma_done);
    call enable_cpu_interrupts;
    do while (copy_ie_register = transmit_dma_done);
    end;   /* loop until the interrupt handler
              takes care of the TDD interrupt -
              it sets COPY_IE_REG = FBA  */

    call perform_command(load_group_addresses);

end program_group_addresses;

/*****************************************************************/


end;   /* system device handler and packet processor */
```

```
*****************************************************************
*****************************************************************
**                    ASMROUT.A86 FILE                         **
*****************************************************************
*****************************************************************

extrn hl_interrupt_handler : far

public write_io_port
public read_io_port
public write_bar
public initialize_cpu_interrupts
public enable_cpu_interrupts
public disable_cpu_interrupts
;****************************************************************


write_io_port:


     ; Parameter Passing Specification:

     ;                      entry                    exit
     ;
     ; parameter 1     <port address>           <unchanged>
     ;
     ; parameter 2     <value to be outputted>  <unchanged>
     ;
     ;

          dseg

          port_address   rb   1

          cseg


     push bx! push si! push dx! push ax
          mov  si, [bx]
          mov  al, [si]
          mov  port_address, al
          mov  si, 2[bx]
          mov  al, [si]
          mov  dl, port_address
          mov  dh, 00h
          out  dx, al
     pop ax! pop dx! pop si! pop bx
          ret

;****************************************************************
```

```
read_io_port:

     ; Parameter Passing Specification
     ;
     ;                         entry                        exit
     ;
     ; parameter 1     <port address>             <unchanged>
     ; parameter 2     <meaningless>          <register value>

          cseg
     push bx! push si! push dx! push ax
          mov  si,  [bx]
          mov  al,  [si]
          mov  port_address,  al
     mov  si, 2[bx]
     mov  dl, port_address
          mov  dh,  00h
          in   al,  dx
          mov  [si], al
     pop  ax! pop dx! pop si! pop bx!
          ret

;*************************************************************************

write_bar:

    ; Parameter Passing Specification
    ;
    ; parameter 1 (and only): the address of the data block
    ;                          to be transmitted or received.

          dseg

          e_bar_port     equ  0b9h
          h_bar_port     equ  0bah
          l_bar_port     equ  0bbh
          temp_e_byte    rb   1
     temp_es         rw   1

          cseg

    ; This module computes a 24 bit address from a 32 bit
    ; address - actually a combination of the ES register
    ; and the IP passed via a parameter list.

     push bx! push ax! push cx! push es! push dx! push si


     mov  dx,  0800h        ; shared memory segment
```

```
    mov  es,  dx
    mov  temp_es, es
    mov  dx,  es
         mov  si,  [bx]
         mov  ax,  [si]
         mov  cl,  12
         shr  dx,  cl
         mov  temp_e_byte,   dl
    mov  dx,  temp_es
         mov  cl,  4
         shl  dx,  cl
         add  ax,  dx
         jnc  no_add
add_1:   inc  temp_e_byte
no_add:  out  l_bar_port, al
         mov  al, ah
         out  h_bar_port, al
         mov  al, temp_e_byte
         out  e_bar_port, al
         pop si! pop dx! pop es! pop cx! pop ax! pop bx
    ret




;------------------------------------ -----------------


initialize_cpu_interrupts:


    ; Module Interface Specification:

    ;     Caller:         Ethertest(PL/I) Procedure

    ;     Parameters:     NONE

    initmodule cseg common
              org 114h
              int5_offset    rw 1
              int5_segment   rw 1

              cseg
              push bx
         push ax
              mov  bx,  offset interrupt_handler
              mov  ax,  0
              push ds
              mov  ds,  ax
```

```
        mov  ds:int5_offset, bx
        mov  bx, cs
        mov  ds:int5_segment, bx
        pop  ds
    pop  ax
        pop  bx
        sti
        ret

;-----------------------------------------------------------

enable_cpu_interrupts:

    ; Module Interface Specification:

    ;     Caller:          Ethertest(PL/I) Procedure

    ;     Parameters:      NONE

        sti
        ret

;-----------------------------------------------------------

disable_cpu_interrupts:

    ; Module Interface Specification:

    ;     Caller:          Ethertest(PL/I) Procedure
    ;     Parameters:      none

        cli
    ret

;-----------------------------------------------------------

interrupt_handler:

    ; IP, CS, and flags are already on stack
    ; save all other registers

    push ax
        push bx
    push cx
    push dx
```

```
        push si
        push di
        push bp
        push ds
        push es
             call hl_interrupt_handler ; high level source
                                       ; routine
        ; restore registers

        pop es
        pop ds
        pop bp
        pop di
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
             sti
             iret

end
```

APPENDIX D
Distributed Decision Algorithm Source Code

PA2, PA3, PB2, and PB3, the distributed user processes which implement the distributed decision algorithm described in Chapter III, are documented herein. Note that the systems file SYSDEF, described in Appendix B, must also be available for compilation of each user process.

Processes PA2 and PA3 are linked as described in Appendix A. Their associated command files NUM12.CMD and NUM13.CMD are loaded into local memory of SBC #2 and SBC #3 respectively in cluster A at runtime. PB2 and PB3 produce NUM22.CMD and NUM23.CMD which are loaded into the memories of cluster B in the same way.

Processes are loaded when requested under MCORTEX control and execution begins and continues until an await state is encountered. Once all processes have been loaded, the various await states will be satisfied by advances of eventcounts in other processes and operation will continue until all input data vectors are processed.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*   PA2 is resident in local memory of SBC 2, CLUSTER A.            */
/*                                                                   */
/*   This procedure performs the following operations:               */
/*                                                                   */
/*      1. Loads quadratic equation parameters A,B,C,D.              */
/*      2. Reads sensor A observation vectors from disk.             */
/*      3. Computes LLR ( LAMBDA_A_X ) for local use.                */
/*      4. Computes ( LAMBDA_A_X + LAMBDA_BP_Y ) the                 */
/*         sum of the local and remote sensor LLR's.                 */
/*      5. Compares the result to the decision threshold             */
/*         and displays the final result and decision.               */
/*      6. Performs steps 2-5 for each input vector.                 */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/

PA2: PROCEDURE;

   %INCLUDE 'SYSDEF.PLI';
   %REPLACE
         PC    BY '8DD0'B4,    /* P3 IS SET TO THIS VALUE */
         TRUE               BY   '1'B,
         FALSE              BY   '0'B,
         ONE                BY   '0001'B4;

   DECLARE
    /*****************************************************************/
    /*    PARFILE CONTAINS THE FOLLOWING PARAMETERS                  */
    /*                                                               */
    /*         1. MATRIX/VECTOR DIMENSION.                           */
    /*         2. D DIAGONAL ELEMENTS OF THE MATRIX-A.               */
    /*         3. COL BY COL ELEMENTS BELOW DIAGONAL OF              */
    /*                                             MATRIX-A.         */
    /*         4. D ELEMENTS OF VECTOR-B.                            */
    /*         5. SCALAR-C.                                          */
    /*         6. THRESHOLD.                                         */
    /*                                                               */
    /*    DATFILE CONTAINS THE FOLLOWING VALUES                      */
    /*                                                               */
    /*         1. D-ELEMENT X-VECTORS.                               */
    /*                                                               */
    /*****************************************************************/

         (PARFILE,DATFILE) FILE,
         EOF BIT(1) STATIC INIT(FALSE),
         (I,J,D,N) FIXED,
       (A(528),B(32),C,T2,X(32),THRESH,LAMBDA_A_X) FLOAT,
         K BIT(16) STATIC INIT('0000'B4),
```

```
/*************************************************************/
/*                                                           */
/*  P3 SET TO PC TO BE ADDED TO SEGMENT ADDR 0800            */
/*                                                           */
/*************************************************************/

       P3 POINTER,

/*************************************************************/
/*                                                           */
/*  BASE LAMBDA_BP_Y  AT P3 = PC (OFFSET ADD TO DATA         */
/*                              SEGMENT = 0800 )             */
/*                                                           */
/*************************************************************/

       LAMBDA_BP_Y FLOAT BASED(P3);

/* SET POINTERS TO VALUES INDICATED IN REPLACE  ABOVE */

   UNSPEC(P3) = PC;

/*************************************************************/
/*                                                           */
/*      INPUT PARAMETERS FROM DISK FILE                      */
/*                                                           */
/*    MATRIX & VECTOR DIMENSION (D = INTEGER)                */
/*                                                           */
/*    CALCULATE N = # OF MATRIX ELEMENTS TO INPUT            */
/*                                                           */
/*    MATRIX-A  (SYMMETRIC)                                  */
/*                                                           */
/*      DIAGONAL   ELEMENTS   FIRST   (# = D)                */
/*      COLUMNS BELOW DIAGONAL NEXT   (# = N-D)              */
/*                                                           */
/*    VECTOR-B  (D ELEMENTS)                                 */
/*                                                           */
/*    SCALAR-C  (1 NUMBER)                                   */
/*                                                           */
/*    THRESHOLD (1 NUMBER)                                   */
/*                                                           */
/*************************************************************/

      OPEN FILE(PARFILE) STREAM INPUT;

         GET FILE(PARFILE) LIST (D);

         N = ((D * D)+D)/2;

         DO I=1 TO N;
            GET FILE(PARFILE) LIST (A(I));
         END;
```

```
          DO I=1 TO D;
             GET FILE(PARFILE) LIST (B(I));
          END;

          GET FILE(PARFILE) LIST (C,THRESH);

     PUT SKIP LIST ('DIMENSION =',D,'THRESHOLD =',THRESH);

/*****************************************************/
/*                                                   */
/*        INPUT AND PROCESS X-VECTORS                */
/*                                                   */
/*****************************************************/

        ON ENDFILE(DATFILE) EOF = TRUE;
        OPEN FILE(DATFILE) STREAM INPUT;

        DO WHILE(EOF = FALSE);

              K = ADD2BIT16(K,ONE);

           PUT SKIP(2);
           DO I=1 TO D;
              GET FILE(DATFILE) LIST (X(I));
              PUT SKIP LIST('X      (',I,') =',X(I));
           END;

/* CALC  LAMBDA_A_X = (X-TRANS)*(A _MATRIX)*(X)           */

        LAMBDA_A_X  = 0;
        DO J=1 TO D-1;
         DO I=J+1 TO D;
          LAMBDA_A_X  = LAMBDA_A_X +( A(I+J+1)*X(I)*X(J));
         END;
        END;

        T2 = 0;
        DO I=1 TO D;
           T2 = T2 + ( A(I)*X(I)*X(I));
        END;
        LAMBDA_A_X  = (2*LAMBDA_A_X ) + T2;

/* ADD LAMBDA_A_X  TO ( B-VECTOR)*(X) & STORE             */

        DO I=1 TO D;
           LAMBDA_A_X  = LAMBDA_A_X  + ( B(I) * X(I));
        END;

/* ADD LAMBDA_A_X  TO C  & STORE IN LAMBDA_A_X            */

        LAMBDA_A_X  = LAMBDA_A_X   + C;
```

```
        /**************************************************************/
        /*                                                            */
        /* AWAIT LAMBDA_BP_Y CALCULATED IN THE OTHER CLUSTER          */
        /*                                                            */
        /**************************************************************/

                    CALL AWAIT(B1EVC,K);

                  PUT SKIP(2) LIST('LAMBDA_A_X  =',LAMBDA_A_X );
                  PUT SKIP LIST   ('LAMBDA_BP_Y =',LAMBDA_BP_Y);

        /**************************************************************/
        /*                                                            */
        /*     ADD THE LAMBDA_BP_Y VALUE RECEIVED FROM                */
        /*     THE OTHER CLUSTER VIA THE ETHERNET TO                  */
        /*     THE LAMBDA_A_X VALUE CALCULATED IN THIS                */
        /*     CLUSTER, AND COMPARE TO THE THRESHOLD.                 */
        /*                                                            */
        /**************************************************************/

           T2 = LAMBDA_A_X + LAMBDA_BP_Y;

           IF (T2 > THRESH) THEN DO;

            PUT SKIP LIST('RESULT      =',T2,'IS > THRESHOLD ');

           END;
           ELSE DO;

            PUT SKIP LIST('RESULT      =',T2,'IS < THRESHOLD ');

           END;

           DO I=0 TO 1000;
                DO J=0 TO 500; /* DELAY LOOP */
                END;
           END;

        /**************************************************************/
        /*                                                            */
        /*  NOTIFY BOARD 3 TO CONTINUE WITH NEXT INPUT VECTOR         */
        /*                                                            */
        /**************************************************************/

                    CALL ADVANCE(A2EVC);

END;          /* END OF DO WHILE (EOF = FALSE) LOOP */

  PUT SKIP(3) LIST('END OF INPUT DATA');

END PA2;
```

```
/********************************************************************/
/********************************************************************/
/*                                                                  */
/*   PA3 is resident in local memory of SBC 3, CLUSTER A.           */
/*                                                                  */
/*   This procedure performs the following operations:              */
/*                                                                  */
/*      1. Loads quadratic equation parameters A,B,C,D.             */
/*      2. Reads sensor A observation vectors from disk.            */
/*      3. Computes the Conditional LLR ( LAMBDA_AP_X )             */
/*         to send to sensor B for further computation.             */
/*      4. Submits a request into the ERB queue to send             */
/*         the CLLR statistic to sensor B.                          */
/*      5. Advances eventcount A1EVC to signal sensor B             */
/*         that its awaited statistic is available.                 */
/*                                                                  */
/********************************************************************/
/********************************************************************/

PA3: PROCEDURE;

   %INCLUDE 'SYSDEF.PLI';
   %REPLACE

          PA   BY '8000'B4,     /* P1 IS SET TO THIS VALUE */
          PB   BY '8002'B4,     /* P2 IS SET TO THIS VALUE */
          ERB_BLOCK_LENGTH      BY   20,/* USED TO CONTROL */
          ERB_BLOCK_LENGTH_M1   BY   19,/* ERB  SIZE       */
          TRUE                  BY   '1'B,
          FALSE                 BY   '0'B,
          ONE                   BY   '0001'B4;

   DECLARE
    /***************************************************************/
    /*    PARAFILE CONTAINS THE FOLLOWING PARAMETERS               */
    /*                                                             */
    /*         1. MATRIX/VECTOR DIMENSION.                         */
    /*         2. D DIAGONAL ELEMENTS OF THE MATRIX-AP.            */
    /*         3. COL BY COL ELEMENTS BELOW DIAGONAL OF            */
    /*                                            MATRIX-AP.       */
    /*         4. D ELEMENTS OF VECTOR-BP.                         */
    /*         5. SCALAR-CP.                                       */
    /*                                                             */
    /*    DATAFILE CONTAINS THE FOLLOWING VALUES                   */
    /*                                                             */
    /*         1. D-ELEMENT X-VECTORS.                             */
    /*                                                             */
    /***************************************************************/

          (PARAFILE,DATAFILE) FILE,
          EOF BIT(1) STATIC INIT(FALSE),
```

```
      (I,J,D,N) FIXED,
      (AP(528),BP(32),CP,T1,X(32)) FLOAT,

/*************************************************************/
/*                                                          */
/*    INDEX VARIABLES AND CONSTANTS USED FØR                */
/*                                                          */
/*          INDEXING IN THE ERB (ERB_INDEX)                 */
/*          SEQUENCING & CONTROL( II,JJ,K )                 */
/*          IDENTIFYING DATA TRANSFER(DATA_TYPE)            */
/*          IDENTIFYING OPPOSITE CLUSTER_ADDRESS            */
/*                                                          */
/*************************************************************/

      ERB_INDEX         FIXED,
      (II,JJ)           BIT(16),
      K                 BIT(16) STATIC INIT('0000'B4),
      DATA_TYPE         BIT(8)  STATIC INIT('01'B4),
      CLUSTER_ADDRESS   BIT(16) STATIC INIT('0002'B4),

/*************************************************************/
/*                                                          */
/*   POINTERS ARE USED IN THE FOLLOWING MANNER              */
/*                                                          */
/*    P1 SET TO PA TO BE ADDED TO SEGMENT ADDR 0800         */
/*    P2 SET TO PB TO BE ADDED TO SEGMENT ADDR 0800         */
/*                                                          */
/*************************************************************/

      (P1,P2) POINTER, .

/*************************************************************/
/*                                                          */
/*       THE ETHERNET REQUEST BLOCK (ERB)                   */
/*                                                          */
/*    ETHERNET REQUEST PACKET (ERP) STRUCTURE               */
/*                                                          */
/*        IS USED IN THE FOLLOWING MANNER                   */
/*                                                          */
/*    COMMAND = 1    FOR DATA TRANSFER OVER E-NET           */
/*    TYPE           (NOT USED BY THIS PROCEDURE)           */
/*    VALUE          (NOT USED BY THIS PROCEDURE)           */
/*    REMOTE_ADDR = CLUSTER_ADDRESS OF DESTINATION          */
/*                                                          */
/*************************************************************/

      1 ERB(0:ERB_BLOCK_LENGTH_M1) BASED (P1),
        2 COMMAND         BIT(8),
        2 TYPE            BIT(8),
        2 VALUE           BIT(16),
        2 REMOTE_ADDR     BIT(16),
```

```
/**************************************************************/
/*                                                            */
/*  BASE LAMBDA_AP_X  AT P2 = PB (OFFSET ADD TO DATA         */
/*                                  SEGMENT = 0800 )          */
/*                                                            */
/**************************************************************/

     LAMBDA_AP_X FLOAT BASED(P2);

/* SET POINTERS TO VALUES INDICATED IN REPLACE  ABOVE */

   UNSPEC(P1) = PA;

   UNSPEC(P2) = PB;

/**************************************************************/
/*                                                            */
/*      INPUT PARAMETERS FROM DISK FILE                       */
/*                                                            */
/*    MATRIX & VECTOR DIMENSION (D = INTEGER)                 */
/*                                                            */
/*    CALCULATE N = # OF MATRIX ELEMENTS TO INPUT             */
/*                                                            */
/*    MATRIX-AP (SYMMETRIC)                                   */
/*                                                            */
/*       DIAGONAL   ELEMENTS   FIRST    (# = D)               */
/*       COLUMNS BELOW DIAGONAL NEXT    (# = N-D)             */
/*                                                            */
/*    VECTOR-BP (D ELEMENTS)                                  */
/*                                                            */
/*    SCALAR-CP (1 NUMBER)                                    */
/*                                                            */
/**************************************************************/

    OPEN FILE(PARAFILE) STREAM INPUT;

       GET FILE(PARAFILE) LIST (D);

       N = ((D * D)+D)/2;

       DO I=1 TO N;
          GET FILE(PARAFILE) LIST (AP(I));
       END;

       DO I=1 TO D;
          GET FILE(PARAFILE) LIST (BP(I));
       END;

       GET FILE(PARAFILE) LIST (CP);

   PUT SKIP LIST ('DIMENSION =',D);
```

```
/*************************************************************/
/*                                                           */
/*        INPUT AND PROCESS X-VECTORS                        */
/*                                                           */
/*************************************************************/

      ON ENDFILE(DATAFILE) EOF = TRUE;

      OPEN FILE(DATAFILE) STREAM INPUT;

      DO WHILE(EOF = FALSE);

         CALL AWAIT(A2EVC,K);

           PUT SKIP(2);

           DO I=1 TO D;

              GET FILE(DATAFILE) LIST (X(I));

              PUT SKIP LIST('X      (',I,') =',X(I));

           END;

/* STORE (X-TRANS)*(AP-MATRIX)*(X) IN LAMBDA_AP_X       */

      LAMBDA_AP_X = 0;

      DO J=1 TO D-1;
       DO I=J+1 TO D;
        LAMBDA_AP_X = LAMBDA_AP_X+(AP(I+J+1)*X(I)*X(J));
       END;
      END;

      T1 = 0;

      DO I=1 TO D;
         T1 = T1 + (AP(I)*X(I)*X(I));
      END;

         LAMBDA_AP_X = (2*LAMBDA_AP_X) + T1;

/* ADD LAMBDA_AP_X TO (BP-VECTOR)*(X) & STORE               */

      DO I=1 TO D;
         LAMBDA_AP_X = LAMBDA_AP_X + (BP(I) * X(I));
      END;

/* ADD LAMBDA_AP_X TO CP & STORE IN LAMBDA_AP_X             */

         LAMBDA_AP_X = LAMBDA_AP_X + CP;
```

```
/*************************************************************/
/*                                                           */
/*     GET A TICKET TO ENABLE A WRITE TO THE ERB             */
/*                                                           */
/*************************************************************/

        II = TICKET(ERB_WRITE_REQUEST);

/* II NOW HAS THE VALUE OF THE TICKET RETURNED               */

        JJ = READ(ERB_WRITE);

/* JJ NOW HAS THE VALUE OF ERB_WRITE                         */

        DO WHILE(JJ < II);

           JJ = READ(ERB_WRITE);

        END;

/* IF ETHERNET REQUEST BLOCK (ERB) IS FULL-BUSY WAIT         */

        JJ = READ(ERB_READ);

        DO WHILE((II - JJ) >= ERB_BLOCK_LENGTH);

           JJ = READ(ERB_READ);

        END;

/*************************************************************/
/*                                                           */
/* WRITE TO ERB WHEN A SLOT IS OPEN                          */
/* COMMAND = 1 FOR DATA TO BE TRANSFERRED                    */
/* REMOTE_ADDR = DESTINATION CLUSTER ADDRESS                 */
/*                                                           */
/*************************************************************/

    ERB_INDEX = MOD(II,ERB_BLOCK_LENGTH);

    ERB(ERB_INDEX).COMMAND = DATA_TYPE;

    ERB(ERB_INDEX).REMOTE_ADDR = CLUSTER_ADDRESS;

/*************************************************************/
/*                                                           */
/*     NOTIFY MCORTEX THAT ERB WRITE IS COMPLETE             */
/*                                                           */
/*************************************************************/

        CALL ADVANCE(ERB_WRITE);
```

```
    /*****************************************************************/
    /*                                                               */
    /*     AN ETHERNET REQUEST PACKET (ERP) IS NOW SETUP             */
    /* IN THE ETHERNET REQUEST BLOCK  (ERB). THIS WILL               */
    /* SIGNAL THE DRIVER PROCEDURE ON BOARD 1 TO FETCH               */
    /* THE DATA STORED IN COMMON MEMORY (LAMBDA_AP_X) AT             */
    /* ADDRESS  0800:8CC0-0800:8CC3 & MOVE IT TO                     */
    /* ADDRESS  0800:80DA-0800:80DD (TRANSMIT_DATA_BLOCK)            */
    /* ALSO IN COMMON MEMORY TO BE PACKETIZED AND SENT TO            */
    /* THE RECEIVE_DATA_BLOCK (RDB) OF THE OTHER CLUSTER             */
    /* ADDRESS  0800:867C-0800:867F WHERE IT IS MOVED TO             */
    /* ADDRESS  0800:8DD0-0800-8DD3 IN THE OTHER CLUSTERS            */
    /* COMMON MEMORY (LAMBDA_AP_X).                                  */
    /*                                                               */
    /*****************************************************************/

    /*****************************************************************/
    /*                                                               */
    /*     NOTIFY OTHER CLUSTER THAT DATA IS READY                   */
    /*                                                               */
    /*****************************************************************/

              CALL ADVANCE(A1EVC);

         K = ADD2BIT16(K,ONE);

END;          /* END OF DO WHILE (EOF = FALSE) LOOP */

  PUT SKIP(3) LIST('END OF INPUT DATA');

END PA3;

/*******************************************************************/
/*******************************************************************/
/*                                                                 */
/*   PB2 is resident in local memory of SBC 2, CLUSTER B.          */
/*                                                                 */
/*   This procedure performs the following operations:             */
/*                                                                 */
/*      1. Loads quadratic equation parameters A,B,C,D.            */
/*      2. Reads sensor B observation vectors from disk.           */
/*      3. Computes LLR ( LAMBDA_B_Y ) for local use.              */
/*      4. Computes ( LAMBDA_B_Y + LAMBDA_AP_X ) the               */
/*         sum of the local and remote sensor LLR's.               */
/*      5. Compares the result to the decision threshold           */
/*         and displays the final result and decision.             */
/*      6. Performs steps 2-5 for each input vector.               */
/*                                                                 */
/*******************************************************************/
/*******************************************************************/
```

```
PB2: PROCEDURE;

   %INCLUDE 'SYSDEF.PLI';
   %REPLACE
           PC    BY '8DD0'B4,      /* P3 IS SET TO THIS VALUE */
           TRUE                    BY   '1'B,
           FALSE                   BY   '0'B,
           ONE                     BY   '0001'B4;

   DECLARE
    /*************************************************************/
    /*    PARFILE CONTAINS THE FOLLOWING PARAMETERS             */
    /*                                                           */
    /*         1. MATRIX/VECTOR DIMENSION.                       */
    /*         2. D DIAGONAL ELEMENTS OF THE MATRIX-A.           */
    /*         3. COL BY COL ELEMENTS BELOW DIAGONAL OF          */
    /*                                          MATRIX-A.        */
    /*         4. D ELEMENTS OF VECTOR-B.                        */
    /*         5. SCALAR-C.                                      */
    /*         6. THRESHOLD.                                     */
    /*                                                           */
    /*    DATFILE CONTAINS THE FOLLOWING VALUES                  */
    /*                                                           */
    /*         1. D-ELEMENT Y-VECTORS.                           */
    /*                                                           */
    /*************************************************************/

           (PARFILE,DATFILE) FILE,
           EOF BIT(1) STATIC INIT(FALSE),
           (I,J,D,N) FIXED,
         (A(528),B(32),C,T2,Y(32),THRESH,LAMBDA_B_Y) FLOAT,
           K BIT(16) STATIC INIT('0000'B4),

    /*************************************************************/
    /*                                                           */
    /*  P3 SET TO PC TO BE ADDED TO SEGMENT ADDR 0800            */
    /*                                                           */
    /*************************************************************/

           P3 POINTER,

    /*************************************************************/
    /*                                                           */
    /*  BASE LAMEDA_AP_X  AT P3 = PC (OFFSET ADD TO DATA         */
    /*                                 SEGMENT = 0800 )          */
    /*                                                           */
    /*************************************************************/

           LAMBDA_AP_X FLOAT BASED(P3);

    /* SET POINTERS TO VALUES INDICATED IN REPLACE  ABOVE */
```

```
   UNSPEC(P3) = PC;

/**************************************************************/
/*                                                            */
/*      INPUT PARAMETERS FROM DISK FILE                      */
/*                                                            */
/*    MATRIX & VECTOR DIMENSION (D = INTEGER)                */
/*                                                            */
/*    CALCULATE N = # OF MATRIX ELEMENTS TO INPUT            */
/*                                                            */
/*    MATRIX-A  (SYMMETRIC)                                  */
/*                                                            */
/*       DIAGONAL   ELEMENTS   FIRST   (# = D)               */
/*       COLUMNS BELOW DIAGONAL NEXT   (# = N-D)             */
/*                                                            */
/*    VECTOR-B  (D ELEMENTS)                                 */
/*                                                            */
/*    SCALAR-C  (1 NUMBER)                                   */
/*                                                            */
/*    THRESHOLD (1 NUMBER)                                   */
/*                                                            */
/**************************************************************/

      OPEN FILE(PARFILE) STREAM INPUT;

         GET FILE(PARFILE) LIST (D);
         N = ((D * D)+D)/2;

         DO I=1 TO N;
            GET FILE(PARFILE) LIST (A(I));
         END;

         DO I=1 TO D;
            GET FILE(PARFILE) LIST (B(I));
         END;

         GET FILE(PARFILE) LIST (C,THRESH);

   PUT SKIP LIST ('DIMENSION =',D,'THRESHOLD =',THRESH);

/**************************************************************/
/*                                                            */
/*       INPUT AND PROCESS Y-VECTORS                         */
/*                                                            */
/**************************************************************/

      ON ENDFILE(DATFILE) EOF = TRUE;

      OPEN FILE(DATFILE) STREAM INPUT;

      DO WHILE(EOF = FALSE);
```

```
                K = ADD2BIT16(K,ONE);

           PUT SKIP(2);
           DO I=1 TO D;
              GET FILE(DATFILE) LIST (Y(I));
              PUT SKIP LIST('Y      (',I,') =',Y(I));
           END;

/* CALC  LAMBDA_B_Y = (Y-TRANS)*(A _MATRIX)*(Y)          */

      LAMBDA_B_Y  = 0;
      DO J=1 TO D-1;
       DO I=J+1 TO D;
        LAMBDA_B_Y  = LAMBDA_B_Y +( A(I+J+1)*Y(I)*Y(J));
       END;
      END;

      T2 = 0;
      DO I=1 TO D;
         T2 = T2 + ( A(I)*Y(I)*Y(I));
      END;

      LAMBDA_B_Y  = (2*LAMBDA_B_Y ) + T2;

/* ADD LAMBDA_B_Y  TO ( B-VECTOR)*(Y) & STORE            */

      DO I=1 TO D;
         LAMBDA_B_Y  = LAMBDA_B_Y  + ( B(I) * Y(I));
      END;

/* ADD LAMBDA_B_Y  TO C  & STORE IN LAMBDA_B_Y           */

      LAMBDA_B_Y  = LAMBDA_B_Y   + C;

/*******************************************************/
/*                                                     */
/* AWAIT LAMBDA_AP_X CALCULATED IN THE OTHER CLUSTER   */
/*                                                     */
/*******************************************************/

            CALL AWAIT(A1EVC,K);
           PUT SKIP(2) LIST('LAMBDA_B_Y  =',LAMBDA_B_Y );
           PUT SKIP LIST   ('LAMBDA_AP_X =',LAMBDA_AP_X);

/*******************************************************/
/*    ADD THE LAMBDA_AP_X VALUE RECEIVED FROM          */
/*    THE OTHER CLUSTER VIA THE ETHERNET TO            */
/*    THE LAMBDA_B_Y VALUE CALCULATED IN THIS          */
/*    CLUSTER, AND COMPARE THE RESULT TO THE           */
/*    THRESHOLD VALUE.                                 */
/*******************************************************/
```

```
      T2 = LAMBDA_B_Y + LAMBDA_AP_X;

     IF (T2 > THRESH) THEN DO;
      PUT SKIP LIST('RESULT        =',T2,'IS > THRESHOLD ');
     END;
     ELSE DO;
      PUT SKIP LIST('RESULT        =',T2,'IS < THRESHOLD ');
     END;

     DO I=0 TO 1000;
          DO J=0 TO 500; /* DELAY LOOP */
          END;
     END;

   /***************************************************************/
   /*                                                             */
   /*  NOTIFY BOARD 3 TO CONTINUE WITH NEXT INPUT VECTOR          */
   /*                                                             */
   /***************************************************************/

              CALL ADVANCE(B2EVC);

END;          /* END OF DO WHILE (EOF = FALSE) LOOP */

  PUT SKIP(3) LIST('END OF INPUT DATA');

END PB2;

/*****************************************************************/
/*****************************************************************/
/*                                                               */
/*   PB3 is resident in local memory of SBC 3, CLUSTER B.        */
/*                                                               */
/*   This procedure performs the following operations:          */
/*                                                               */
/*      1. Loads quadratic equation parameters A,B,C,D.          */
/*      2. Reads sensor B observation vectors from disk.         */
/*      3. Computes the Conditional LLR ( LAMBDA_BP_Y )          */
/*         to send to sensor A for further computation.          */
/*      4. Submits a request into the FRB queue to send          */
/*         the CLLR statistic to sensor A.                       */
/*      5. Advances eventcount B1EVC to signal sensor A          */
/*         that its awaited statistic is available.              */
/*                                                               */
/*****************************************************************/
/*****************************************************************/

PB3: PROCEDURE;

  %INCLUDE 'SYSDEF.PLI';

  %REPLACE
```

```
        PA    BY '8000'B4,    /* P1 IS SET TO THIS VALUE */
        PB    BY '8CC0'B4,    /* P2 IS SET TO THIS VALUE */

        ERB_BLOCK_LENGTH     BY   20,/* USED TO CONTROL */
        ERB_BLOCK_LENGTH_M1  BY   19,/* ERB  SIZE       */

        TRUE                 BY   '1'B,
        FALSE                BY   '0'B,

        ONE                  BY   '0001'B4;

DECLARE


 /*********************************************************/
 /*    PARAFILE CONTAINS THE FOLLOWING PARAMETERS         */
 /*                                                       */
 /*         1. MATRIX/VECTOR DIMENSION.                   */
 /*         2. D DIAGONAL ELEMENTS OF THE MATRIX-AP.      */
 /*         3. COL BY COL ELEMENTS BELOW DIAGONAL OF      */
 /*                                      MATRIX-AP.       */
 /*         4. D ELEMENTS OF VECTOR-BP.                   */
 /*         5. SCALAR-CP.                                 */
 /*                                                       */
 /*    DATAFILE CONTAINS THE FOLLOWING VALUES             */
 /*                                                       */
 /*         1. D-ELEMENT Y-VECTORS.                       */
 /*                                                       */
 /*********************************************************/

        (PARAFILE,DATAFILE) FILE,
        EOF BIT(1) STATIC INIT(FALSE),
        (I,J,D,N) FIXED,
        (AP(528),BP(32),CP,T1,Y(32)) FLOAT,

 /*********************************************************/
 /*                                                       */
 /*    INDEX VARIABLES AND CONSTANTS USED FOR             */
 /*                                                       */
 /*          INDEXING IN THE ERB (ERB_INDEX)              */
 /*          SEQUENCING & CONTROL( II,JJ,K )              */
 /*          IDENTIFYING DATA TRANSFER(DATA_TYPE)         */
 /*          IDENTIFYING OPPOSITE CLUSTER_ADDRESS         */
 /*                                                       */
 /*********************************************************/

        ERB_INDEX         FIXED,
        (II,JJ)           BIT(16),
        K                 BIT(16) STATIC INIT('0000'B4),
        DATA_TYPE         BIT(8)  STATIC INIT('01'B4),
        CLUSTER_ADDRESS   BIT(16) STATIC INIT('0001'B4),
```

```
/*************************************************************/
/*                                                           */
/*   POINTERS ARE USED IN THE FOLLOWING MANNER             . */
/*                                                           */
/*    P1 SET TO PA TO BE ADDED TO SEGMENT ADDR 0800          */
/*    P2 SET TO PB TO BE ADDED TO SEGMENT ADDR 0800          */
/*                                                           */
/*************************************************************/


      (P1,P2) POINTER,


/*************************************************************/
/*                                                           */
/*      THE ETHERNET REQUEST BLOCK (ERB)                     */
/*                                                           */
/*    ETHERNET REQUEST PACKET (ERP) STRUCTURE                */
/*                                                           */
/*       IS USED IN THE FOLLOWING MANNER                     */
/*                                                           */
/*    COMMAND = 1    FOR DATA TRANSFER OVER E-NET            */
/*    TYPE           (NOT USED BY THIS PROCEDURE)            */
/*    VALUE          (NOT USED BY THIS PROCEDURE)            */
/*    REMOTE_ADDR = CLUSTER_ADDRESS OF DESTINATION           */
/*                                                           */
/*************************************************************/


      1 ERB(0:ERB_BLOCK_LENGTH_M1) BASED (P1),
        2 COMMAND          BIT(8),
        2 TYPE             BIT(8),
        2 VALUE            BIT(16),
        2 REMOTE_ADDR      BIT(16),


/*************************************************************/
/*                                                           */
/*  BASE LAMBDA_BP_Y  AT P2 = PB (OFFSET ADD TO DATA         */
/*                               SEGMENT = 0800 )            */
/*                                                           */
/*************************************************************/


      LAMBDA_BP_Y FLOAT BASED(P2);

/* SET POINTERS TO VALUES INDICATED IN REPLACE  ABOVE */

   UNSPEC(P1) = PA;

   UNSPEC(P2) = PB;
```

```
/*****************************************************************/
/*                                                               */
/*      INPUT PARAMETERS FROM DISK FILE                          */
/*                                                               */
/*    MATRIX & VECTOR DIMENSION (D = INTEGER)                    */
/*                                                               */
/*    CALCULATE N = # OF MATRIX ELEMENTS TO INPUT                */
/*                                                               */
/*    MATRIX-AP (SYMMETRIC)                                      */
/*                                                               */
/*       DIAGONAL   ELEMENTS   FIRST   (# = D)                   */
/*       COLUMNS BELOW DIAGONAL NEXT   (# = N-D)                 */
/*                                                               */
/*    VECTOR-BP (D ELEMENTS)                                     */
/*                                                               */
/*    SCALAR-CP (1 NUMBER)                                       */
/*                                                               */
/*****************************************************************/

    OPEN FILE(PARAFILE) STREAM INPUT;

       GET FILE(PARAFILE) LIST (D);
       N = ((D * D)+D)/2;

       DO I=1 TO N;
          GET FILE(PARAFILE) LIST (AP(I));
       END;

       DO I=1 TO D;
          GET FILE(PARAFILE) LIST (BP(I));
       END;

       GET FILE(PARAFILE) LIST (CP);

  PUT SKIP LIST ('DIMENSION =',D);

/*****************************************************************/
/*                                                               */
/*       INPUT AND PROCESS Y-VECTORS                             */
/*                                                               */
/*****************************************************************/

    ON ENDFILE(DATAFILE) EOF = TRUE;

    OPEN FILE(DATAFILE) STREAM INPUT;

    DO WHILE(EOF = FALSE);

       CALL AWAIT(B2EVC,K);

         PUT SKIP(2);
```

```
          DO I=1 TO D;
             GET FILE(DATAFILE) LIST (Y(I));
             PUT SKIP LIST('Y     (',I,') =',Y(I));
          END;

/* STORE (Y-TRANS)*(AP-MATRIX)*(Y) IN LAMBDA_BP_Y       */

      LAMBDA_BP_Y = 0;
      DO J=1 TO D-1;
       DO I=J+1 TO D;
        LAMBDA_BP_Y = LAMBDA_BP_Y+(AP(I+J+1)*Y(I)*Y(J));
       END;
      END;

      T1 = 0;

      DO I=1 TO D;
         T1 = T1 + (AP(I)*Y(I)*Y(I));
      END;

         LAMBDA_BP_Y = (2*LAMBDA_BP_Y) + T1;

/* ADD LAMBDA_BP_Y TO (BP-VECTOR)*(Y) & STORE           */

      DO I=1 TO D;
         LAMBDA_BP_Y = LAMBDA_BP_Y + (BP(I) * Y(I));
      END;

/* ADD LAMBDA_BP_Y TO CP & STORE IN LAMBDA_BP_Y         */

         LAMBDA_BP_Y = LAMBDA_BP_Y + CP;

/*********************************************************/
/*                                                       */
/*     GET A TICKET TO ENABLE A WRITE TO THE ERB         */
/*                                                       */
/*********************************************************/

            II = TICKET(ERB_WRITE_REQUEST);

/* II NOW HAS THE VALUE OF THE TICKET RETURNED           */

            JJ = READ(ERB_WRITE);

/* JJ NOW HAS THE VALUE OF ERB_WRITE                     */

            DO WHILE(JJ < II);

               JJ = READ(ERB_WRITE);

            END;
```

```
/* IF ETHERNET REQUEST BLOCK (ERB) IS FULL-BUSY WAIT     */

            JJ = READ(ERB_READ);

            DO WHILE((II - JJ) >= ERB_BLOCK_LENGTH);

               JJ = READ(ERB_READ);

            END;

/*********************************************************/
/*                                                       */
/* WRITE TO ERB WHEN A SLOT IS OPEN                      */
/* COMMAND = 1 FOR DATA TO BE TRANSFERED                 */
/* REMOTE_ADDR = DESTINATION CLUSTER ADDRESS             */
/*                                                       */
/*********************************************************/

      ERB_INDEX = MOD(II,ERB_BLOCK_LENGTH);
      ERB(ERB_INDEX).COMMAND = DATA_TYPE;
      ERB(ERB_INDEX).REMOTE_ADDR = CLUSTER_ADDRESS;

/*********************************************************/
/*                                                       */
/*      NOTIFY MCORTEX THAT ERB WRITE IS COMPLETE        */
/*                                                       */
/*********************************************************/

            CALL ADVANCE(ERB_WRITE);

/*********************************************************/
/*                                                       */
/*      AN ETHERNET REQUEST PACKET (ERP) IS NOW SETUP    */
/* IN THE ETHERNET REQUEST BLOCK  (ERB). THIS WILL       */
/* SIGNAL THE DRIVER PROCEDURE ON BOARD 1 TO FETCH       */
/* THE DATA STORED IN COMMON MEMORY (LAMBDA_BP_Y) AT     */
/* ADDRESS  0800:8CC0-0800:8CC3 & MOVE IT TO             */
/* ADDRESS  0800:80DA-0800:80DD (TRANSMIT_DATA_BLOCK)    */
/* ALSO IN COMMON MEMORY TO BE PACKETIZED AND SENT TO    */
/* THE RECEIVE_DATA_BLOCK (RDB) OF THE OTHER CLUSTER     */
/* ADDRESS  0800:967C-0800:967F WHERE IT IS MOVED TO     */
/* ADDRESS  0800:8DD0-0800-8DD3 IN THE OTHER CLUSTERS    */
/* COMMON MEMORY (LAMBDA_BP_Y).                          */
/*                                                       */
/*********************************************************/

/*********************************************************/
/*                                                       */
/*      NOTIFY OTHER CLUSTER THAT DATA IS READY          */
/*                                                       */
/*********************************************************/
```

```
                CALL ADVANCE(B1EVC);
          K = ADD2BIT16(K,ONE);
END;          /* END OF DO WHILE (EOF = FALSE) LOOP */
  PUT SKIP(3) LIST('END OF INPUT DATA');
END PB3;
```

LIST OF REFERENCES

1. Hahn, S.C., Analysis of a Distributed Decision Algorithm, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1985.

2. Brewer, D.J., A Real–Time Executive for Multiple–Computer Clusters, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1984.

3. Tenney, R.R. and Sandell, N.R., "Detection with Distributed Sensors," IEEE Trans. Aerospace and Electronic Systems, Vol. AES-17, No.4, July 1981.

4. Rosenfeld, A., Hummel, R.A. and Zucker, S.W., "Scene Labeling by Relaxation Operations", IEEE Trans. Systems, Man, and Cybernetics, Vol. 6, 1976 pp 420-433.

5. Haralick, R.M., "Decision Making in Context", IEEE Trans. Pattern Anal. Machine Intelligence, Vol. PAMI-5, No.4, July 1983.

6. Van Trees, H.L., Detection Estimation and Modulation Theory, Part I, John Wiley & Sons, New York, 1968.

7. Duda, R.O. and Hart, P.E., Pattern Classification and Scene Analysis, John Wiley & Sons, New York, 1973.

8. Fukunaga, K., Introduction to Statistical Pattern Recognition, Academic Press, New York, 1972.

9. Helstrom, C.W., Probability and Stochastic Processes for Engineers, Macmillan Publishing Company, New York, 1984.

10. Xerox Corporation, The Ethernet – A Local Area Network: Data Link Layer and Physical Layer Specifications, Version 1.0, September 1980.

11. Reed, D.P. and Kanodia, R.J., "Synchronization with Eventcounts and Sequencers", Communication of the ACM, Volume 22, pp. 115-123, February 1979.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5100	2
3.	Department Chairman, Code 62 Electrical and Computer Engineering Dept. Naval Postgraduate School Monterey, California 93943-5100	1
4.	Professor Charles W. Therrien, Code 62Ti Electrical and Computer Engineering Dept. Naval Postgraduate School Monterey, California 93943-5100	3
5.	Professor Uno R. Kodres, Code 52Kr Computer Science Department Naval Postgraduate School Monterey, California 93943-5100	2
6.	Professor M. L. Cotton, Code 62Cc Electrical and Computer Engineering Dept. Naval Postgraduate School Monterey, California 93943-5100	1
7.	Professor R. Panholzer, Code 62Pz Electrical and Computer Engineering Dept. Naval Postgraduate School Monterey, California 93943-5100	1
8.	Capt. Mark A. Schon 1801 Artillery Ridge Road Fredericksburg, Virginia 22401	3

9. Daniel Green, Code 20E 1
Naval Surface Weapons Center
Dahlgren, Virginia 22449

10. Capt. J. Donegan, USN 1
PMS 400B5
Naval Sea System Command
Washington, D.C. 20362

11. RCA AEGIS Depository 1
RCA Corporation
Government Systems Division
Mail Stop 127-327
Moorestown, New Jersey 08057

12. Library, Code E33-05 1
Naval Surface Weapons Center
Dahlgren, Virginia 22449

13. Dr. M.J. Gralia 1
Applied Physics Laboratory
John Hopkins Road
Laurel, Maryland 20707

14. Dana Small 1
Code 8242, NOSC
San Diego, California 92152

15. Lt. Robin K. Weinhold-Schon 1
4582 Marlwood Way
Virginia Beach, Virginia 23462

16. Susan M. Schon 1
Route 1, Box 531
Beaufort, North Carolina 28516

17. Otto W. Schon 1
509 Stafford Avenue
Bristol, Connecticut 06010

18.	Kevin and Michael Schon 1914 Nelson Avenue Memphis, Tennessee 38104	2
19.	Capt. "Wild Bill" Johnson 2708 Pine Manor Lane Albany, Georgia 31707	1
20.	Professor P. Moose, Code 62Me Electrical and Computer Engineering Dept. Naval Postgraduate School Monterey, California 93943-5100	1
21.	Mr. D. Cowan NWC China Lake, Code 31507 China Lake, California 93555	1
22.	Dr. P. Krueger AIR 320H Naval Air Systems Command Washington, D.C. 20361	1

215083

Thesis
S341422 Schon
c.1
Development of a testbed for multi-sensor distributed decision algorithms.